NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

ELINT SIGNAL PROCESSING USING CHOI-WILLIAMS DISTRIBUTION ON RECONFIGURABLE COMPUTERS FOR DETECTION AND CLASSIFICATION OF LPI EMITTERS

by

Teresa Lynn Odom Upperman

March 2008

Thesis Advisor:	Douglas J. Fouts
Co-Advisor:	Phillip E. Pace

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** March 2008	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE ELINT Signal Processing Using Choi-Williams Distribution on Reconfigurable Computers for Detection and Classification of LPI Emitters		**5. FUNDING NUMBERS**	
6. AUTHOR(S) Teresa Lynn Odom Upperman			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Center for Joint Services Electronic Warfare Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Security Agency, Fort Meade, MD; Office of Naval Research, Arlington, VA		**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited		**12b. DISTRIBUTION CODE**	

13. ABSTRACT (maximum 200 words)

This thesis documents the use of the SRC-6 Reconfigurable Computer for use in analyzing low probability of intercept (LPI) signals using the Choi-Williams distribution. The SRC-6 is a reconfigurable computer manufactured by SRC Computers, Inc. which allows the user to tailor both the software and the hardware to a specific task. This increases the speed at which the task can be accomplished making it useful for applications in electronic intelligence (ELINT). The Choi-Williams distribution is a mathematical technique that was first created using MATLAB and then converted to C code for use on the SRC-6. The purpose of this study is to investigate the feasibility of using a reconfigurable computer for ELINT applications and the timely detection and classification of LPI signals. This thesis is part of a larger study to use reconfigurable computers for the autonomous detection and classification of LPI signals.

14. SUBJECT TERMS Choi-Williams Distribution, Reconfigurable Computer, Signal Processing. MATLAB programming, C programming, Low Probability of Intercept (LPI), Radar detection, Radar classification			**15. NUMBER OF PAGES** 104 **16. PRICE CODE**
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

ELINT SIGNAL PROCESSING ON RECONFIGURABLE COMPUTERS FOR DETECTION AND CLASSIFICATION OF LPI EMITTERS

Teresa L.O. Upperman
Civilian, Department of Defense
B.S., Valparaiso University, 2001

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 2008

Author: Teresa Lynn Odom Upperman

Approved by: Douglas J. Fouts
Thesis Advisor

Phillip E. Pace
Co-Advisor

Jeffrey B. Knorr
Chairman, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis documents the use of the SRC-6 Reconfigurable Computer for use in analyzing low probability of intercept (LPI) signals using the Choi-Williams distribution. The SRC-6 is a reconfigurable computer manufactured by SRC Computers, Inc. which allows the user to tailor both the software and the hardware to a specific task. This increases the speed at which the task can be accomplished making it useful for applications in electronic intelligence (ELINT). The Choi-Williams distribution is a mathematical technique that was first created using MATLAB and then converted to C code for use on the SRC-6. The purpose of this study is to investigate the feasibility of using a reconfigurable computer for ELINT applications and the timely detection and classification of LPI signals. This thesis is part of a larger study to use reconfigurable computers for the autonomous detection and classification of LPI signals.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ABBREVIATIONS, ACRONYMS, AND SYMBOLS

ADC	Analog to Digital Converter
CWD	Choi-Williams Distribution
DFT	Discrete Fourier Transform
DMA	Direct Memory Access
DOD	Department of Defense
ELINT	Electronic Intelligence
ΔF	Modulation Bandwidth
FFT	Fast Fourier Transform
FMCW	Frequency Modulated Continuous Wave
FPGA	Field Programmable Gate Arrays
I	In Phase Portion of Data
LPI	Low Probability of Intercept
MAP	Multi-Adaptive Processing
N	Number of Samples
NPS	Naval Postgraduate School
PRI	Pulse Repetition Interval
Q	Quadrature Portion of Data
QMFB	Quadrature Mirror Filter Bank
SNR	Signal To Noise Ratio
USMC	United States Marine Corps
USNR	United States Naval Reserve
WVD	Wigner-Ville Distribution

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

Many thanks to David Caliga of SRC Computer for his quick responses to all questions and his timely supply of SRC Fast Fourier Transform (FFT) MAP code. The documentation he provided of the SRC-6 programming environment along with the documentation on the FFT code was invaluable and made programming on the SRC-6 possible.

I am also thankful to Professor Phillip Pace of NPS who provided oversight over the creation and verification of the Choi-Williams distribution in MATLAB. His research on LPI signal processing using the Wigner-Ville distribution was directly applicable to the generation of the Choi-Williams code.

I would also like to thank Professor Douglas Fouts of NPS who was my thesis advisor and provided the support on SRC-6 questions and directed all of my questions to the best sources of information. His timely suggestions made the thesis process proceed smoothly.

Many thanks also to Dan Zulaica who went above and beyond to install the FFT MAP code as soon as it was received. He also maintained the SRC-6 so that it was always operational when needed and provided documentation on how to use the hardware. He was always available to answer questions no matter how small.

Much thanks goes to our sponsors. This research was supported in part by the National Security Agency for providing the SRC-6 and the Office of Naval Research, Code 312 Arlington, VA, for supporting the algorithm development.

I am also thankful for the support of my husband, who had his own thesis to accomplish and my daughter who was born during our time here at Monterey. However, none of it would have been possible without the on-site support of my father who made it possible to be both new parents and complete our thesis work.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Low probability of intercept (LPI) signals are increasingly difficult to detect using traditional electronic intelligence (ELINT) approaches. Intercepting these signals and extracting parametric data for classification has become more and more challenging.

A possible solution to the detection and classification of LPI signals for ELINT is to use multiple detection strategies against an LPI signal and automate the classification process. This would cover a variety of LPI signals and utilize the strengths of different detection strategies. The drawback to this solution is that many of these strategies take long computational times.

The objective of this thesis was to investigate the feasibility of using the SRC-6 reconfigurable computer to utilize the Choi-Williams distribution (CWD) as a detection strategy. Using a reconfigurable computer may decrease the computational time of the CWD. This objective is part of a larger study to use the SRC-6 for the autonomous detection and classification of LPI signals.

First, the Choi-Williams distribution was designed in MATLAB to validate the strategy against several different types of LPI signals. This was used as a benchmark for future work. Second, the software was converted from MATLAB to standard C code. At this stage the code was functional on the common memory of the SRC but did not utilize the reconfigurable aspect of the SRC-6 hardware. Last, the code was modified to run on the Multi-Adaptive Processing (MAP) board utilizing the reconfigurable hardware aspects of the SRC-6.

The results of the timing analysis show that the C code was the fastest implementation. The MATLAB implementation was highly dependent on the specific computer running the software and any background processes that may have been running. The SRC-6 C code implementation using the Multi-Adaptive Processing (MAP) board was comparable to the MATLAB code run on a 1.6 GHz processor with one gigabyte of random access memory using MATLAB Student Version 7.1. The SRC-6 C

code implantation was faster than the MATLAB code run on a 3 GHz processor with 512 megabytes of random access memory using MATLAB 7.4.0 R2007b.

It should be noted that: (1) the MATLAB timing was greatly dependent on the specific computer running the software and any background processes that may have been running, (2) the Choi-Williams distribution kernel function was the most time consuming portion of the code in all three coding implementations (2) the SRC-6 code can be further optimized for this application and as an ELINT detection system as a whole. Further investigation into optimizing the SRC-6 C code for the Choi-Williams distribution should be accomplished.

I. INTRODUCTION

A. BACKGROUND

Low probability of intercept (LPI) signals are signals that are increasingly difficult to detect using traditional approaches. These signals are commonly mistaken for noise if they are noticed by the detection system at all. LPI signals use a variety of techniques using frequency modulation, phase modulation or a combination of techniques to avoid detection.

Intercepting these signals and extracting parametric data for classification has emerged as a new field of electronic warfare study in recent years. Several strategies for detecting and classifying these signals have emerged. Each strategy has its own strengths and weaknesses against the variety of LPI signals present. Some strategies work well against frequency modulated continuous wave (FMCW) LPI signals and others are used for extracting parameters from polyphase signals.

A possible solution to the detection and classification of LPI signals is to use multiple detection strategies against an LPI signal to extract as much information from the signal as possible. This would cover a variety of LPI signals and utilize the strengths of different detection strategies and compensate where a strategy was weak. The drawback to this solution is that many of these strategies take long computational times.

The ideal implementation of this solution would be to achieve real time parallel processing of several detection strategies. The output of these detection strategies could be run through an automated detection system. This would allow for automated detection, classification, and parameter extraction of the LPI signal modulations.

This thesis is an extension of a larger effort to create an autonomous LPI detection and classification system. This thesis explores the use of the SRC Computers Corporation SRC-6 reconfigurable computer in an automated LPI detection and classification system. Specifically, the SRC computer will be configured to use the Choi-Williams distribution (CWD) as a detection strategy to analyze signals [1].

B. OBJECTIVE

The objective of this thesis was to investigate the feasibility of using the SRC-6 reconfigurable computer to utilize the Choi-Williams distribution for detection and classification of LPI signals. This objective is part of a larger study to use the SRC-6 for the autonomous detection and classification of LPI signals.

C. RELATED WORK

The overall effort of using the SRC-6 for autonomous detection and classification of LPI signals has been the focus of several Naval Postgraduate School (NPS) theses. Figure 1 shows the overall design of the autonomous system.

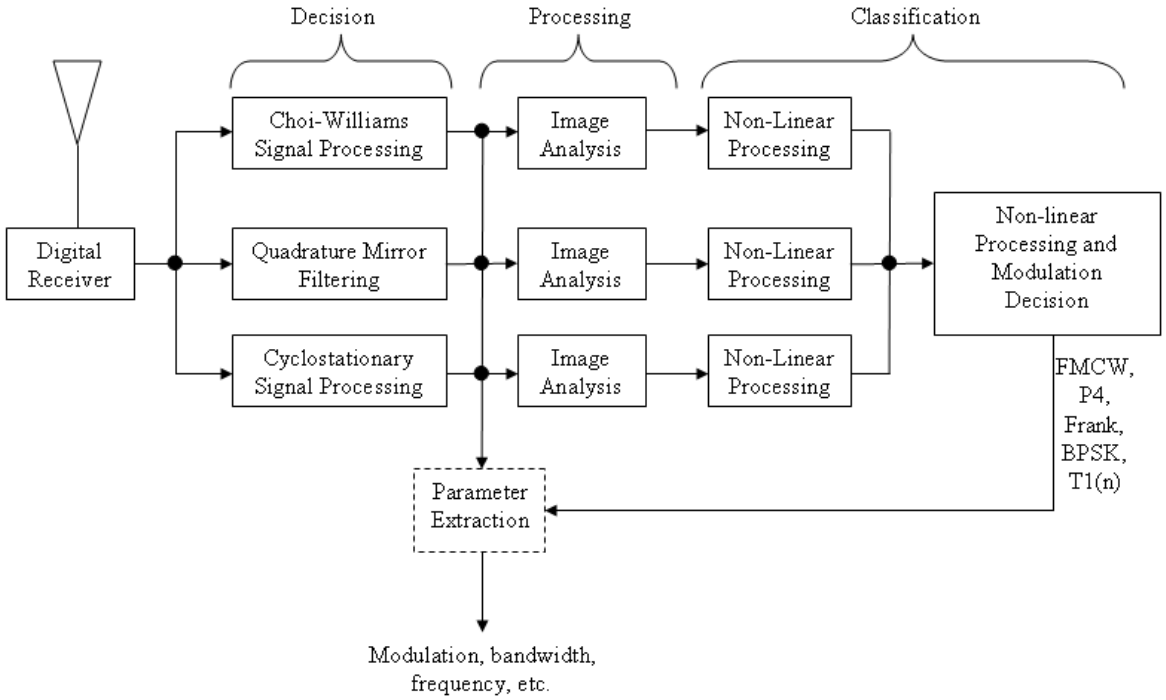

Figure 1: Autonomous Low Probability of Intercept (LPI) Detection and Classification System Using the SRC-6 Reconfigurable Computer. (From: [2])

Captain Kevin Stoffel, United States Marine Corps (USMC) conducted the study of the quadrature mirror filter bank (QMFB) for use on the SRC-6 [2]. This analysis involved the conversion of a detected signal space into a frequency-time plot using

analog to digital (ADC) converters on the SRC-6. Ensign Dane Brown, United States Naval Reserve (USNR), generated the signal processing to convert the frequency-time plot data into a bitmap [3]. This bitmap was used for LPI emitter classifications. Mr. Scott Bailey demonstrated the use of neural networks for the classification of LPI emitters, specifically for use with the QMFB bitmaps [4].

Current work on the LPI detection system, excluding this thesis, include the study of cyclostationary signal processing on the SRC-6 by Gary Upperman, Department of Defense (DOD) civilian[5]. Future thesis work will concentrate on other areas of the LPI detection system as well as the integration of the whole system.

D. THESIS ORGANIZATION

The remainder of this thesis will be organized as follows:

- Chapter II discusses the software development, giving a general background and a verification of the software code.
- Chapter III discusses porting the software to the SRC-6, giving a general background on the SRC-6 hardware and showing the software modifications needed to use the SRC-6 hardware.
- Chapter IV discusses performance analysis of the system using the generated code.
- Chapter V provides an overall summary of the results, the conclusions generated from the results and any future work.

THIS PAGE INTENTIONALLY LEFT BLANK

II. SOFTWARE GENERATION

A. BACKGROUND

Before any code was generated, an analysis of the Choi-Williams Distribution (CWD) was conducted. This distribution is similar to the Wigner-Ville Distribution (WVD) with a exponential kernel. A brief explanation of the distribution can be found in this chapter.

To benchmark any code generated, the LPI Toolbox was used to create LPI signals. This software was developed by Dr. Phillip E. Pace and is documented in [6]. For more information on the LPI signals used, please see Appendix A.

1. Choi-Williams Distribution (CWD)

The Choi-Williams distribution (CWD) is a time-frequency analysis technique used for signal processing. A signal is considered stationary over a short time period and a "snapshot" is taken. A Fourier transform of the sampled in-phase and quadrature data is used to determine the energy distribution of the signal at the time of the sample. This energy distribution can be used to determine signal characteristics. The time period chosen for the sampled data determines the resolution of the Fourier transform. A short period gives poor frequency resolution and a long period in effect blurs the "snapshot".

Several techniques have been identified to increase both frequency and time resolution concurrently. These techniques, or distributions, utilize different kernel functions to achieve better resolution. A kernel is a weighting function applied to the data. The output of a distribution is the original "snapshot" where the data has been smoothed and weighted to increase the time and frequency resolution. From these results signal parameters such as carrier frequency, pulse repetition intervals, coding schemes, etc. can be determined.

One of the most common distributions is the Wigner-Ville distribution which has a kernel of one. This distribution works well for "signals whose instantaneous frequency

and group delay correspond to the same curve in the time-frequency plane" [1]. However, when the WVD is applied to signals that do not fall into this category, there are cross terms that can obscure the results. The CWD described below has an exponential kernel and is useful for reducing the magnitude of the cross terms.

The Choi-Williams distribution was identified by Choi and Williams in 1989 and found in [1]. The equation for the continuous Choi-Williams distribution of the input signal $x(t)$ is given by

$$CWD_x(t,\omega) = \int_{\tau=-\infty}^{\infty} e^{-j\omega\tau} \int_{\mu=-\infty}^{\infty} \frac{1}{\sqrt{4\pi\tau^2/\sigma}} e^{-\frac{(\mu-t)^2}{4\tau^2/\sigma}} x(\mu+\frac{\tau}{2}) x^*(\mu-\frac{\tau}{2}) d\mu d\tau \qquad (1.1)$$

where t is the time variable, ω is the angular frequency variable, σ is a positive-valued scaling factor, and * indicates the complex conjugate. The CWD can also be defined from the Fourier transform $X(\omega)$ of $x(t)$ as shown below.

$$CWD_X(\omega,t) = \frac{1}{2\pi} \int_{\omega_0=-\infty}^{\infty} e^{-j\omega_0 t} \int_{\mu=-\infty}^{\infty} \frac{1}{\sqrt{4\pi\omega_0^2/\sigma}} e^{-\frac{(\mu-\omega)^2}{4\omega_0^2/\sigma}} X(\mu+\frac{\omega_0}{2}) X^*(\mu-\frac{\omega_0}{2}) d\mu d\omega_0 \quad (1.2)$$

This can then be expressed with a discrete time index and windowed for large data sample sets shown in the following equation

$$CWD_x(\ell,\omega) = 2W(n) \sum_{n=-\infty}^{\infty} S(\ell,n) e^{-j2\omega n} \qquad (1.3)$$

where

$$S(\ell,n) = W(\mu) \sum_{\mu=-M/2}^{M/2} \frac{1}{\sqrt{4\pi n^2/\sigma}} e^{-\frac{(\mu-\ell)^2}{4n^2/\sigma}} x(\mu+n) x^*(\mu-n) \qquad (1.4)$$

and $W(n)$ is a symmetrical window (such as Hamming) which has nonzero values on the interval $-N/2$ to $N/2$ and $W(\mu)$ is a uniform rectangular window that has a value of one for the range of $-M/2$ and $M/2$. The choices of N and M on these windows respectively determine the frequency resolution of the CWD and the range at which the function will be defined.

The discrete CWD can be modified to fit the standard discrete Fourier Transform (DFT) by setting $\omega = \pi k/2N$ [7]. The final equation is written

$$CWD_x(\ell, \frac{\pi k}{2N}) = 2\sum_{n=0}^{2N-1} S'(\ell, n)e^{-j2\pi kn/N} \tag{1.5}$$

where the kernel function $S'(\ell, n)$ is defined as

$$S'(\ell, n) = \begin{cases} S(\ell, n), & 0 \le n \le N-1 \\ 0, & n = N \\ S(\ell, n-2N), & N+1 \le n \le 2N-1 \end{cases} \tag{1.6}$$

and $S(\ell, n)$ is defined in Equation 1.4.

2. Choi-Williams Distribution MATLAB Benchmark

Equations (1.5) and (1.6) were then modeled in MATLAB. The MATLAB code was used to benchmark results found on the SRC-6 computer. A full description of the generation of the CWD expression used for all coding in this thesis can be found in [8]. Additional information in [9] and [10] were used to generate these results.

The MATLAB code generates an N by N matrix which results in a three dimensional image that can be used to determine signal characteristics such as frequency, pulse repetition interval (PRI), and signal period to name a few. The variable N is defined as the number of samples input into the system. Due to the repetitive use of the Fast Fourier Transform (FFT) in the software, the size of N is generally set at a small power of two, either 256 or 512 samples. Examples of the image output for N equal to 512 are shown in Figures 2 through 5.

In this example, an LPI signal using FMCW techniques can be seen. The FMCW signal was generated with a carrier frequency of 1 kHz, a sampling frequency of 7 kHz, a modulation frequency (bandwidth) of 250 Hz, a modulation period of 20 μs and a signal to noise ratio (SNR) of 0 dB.

Figure 2 shows the 3-D plot of the software results. Notice that the FMCW signal is clearly visible against the background noise. More information can be seen by examining this graph from different perspectives.

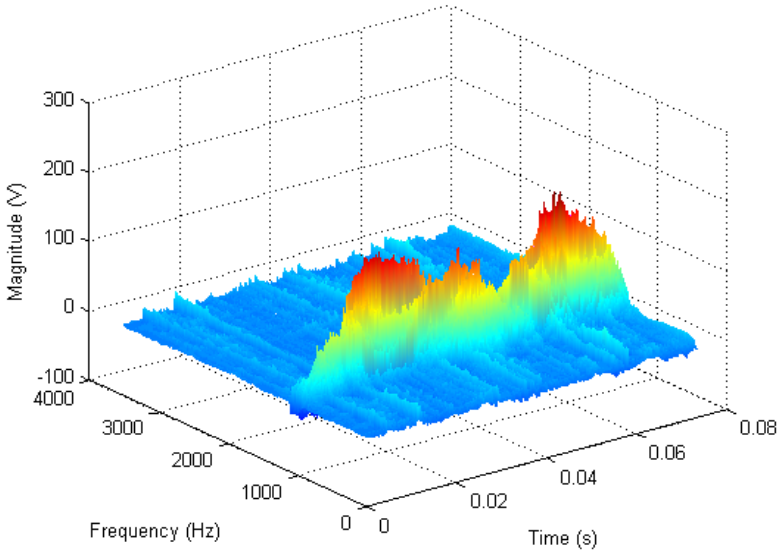

Figure 2: 3-D Mesh Plot of an FMCW Test Signal Using MATLAB CWD Software.

Figure 3 shows a plot of the marginal frequency distribution. This shows the CWD identified the carrier frequency of 1 kHz. The width of the modulation is the FMCW modulation bandwidth, ΔF , of 250 Hz.

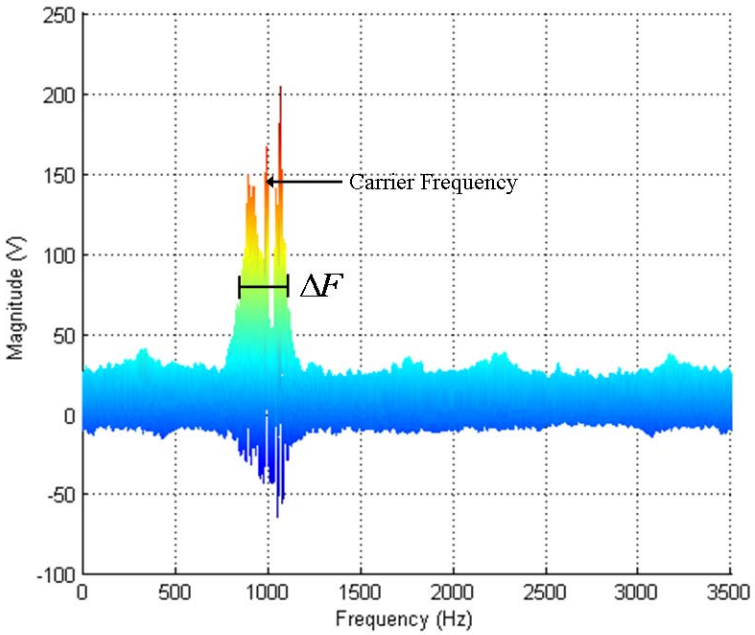

Figure 3: Marginal Frequency Distribution of an FMCW Test Signal Using MATLAB CWD Software.

Figure 4 shows the marginal time distribution. This perspective of an FMCW signal does not clearly show the important characteristics of the signal. It is included here as an example of the type of output available when performing a CWD on an LPI signal.

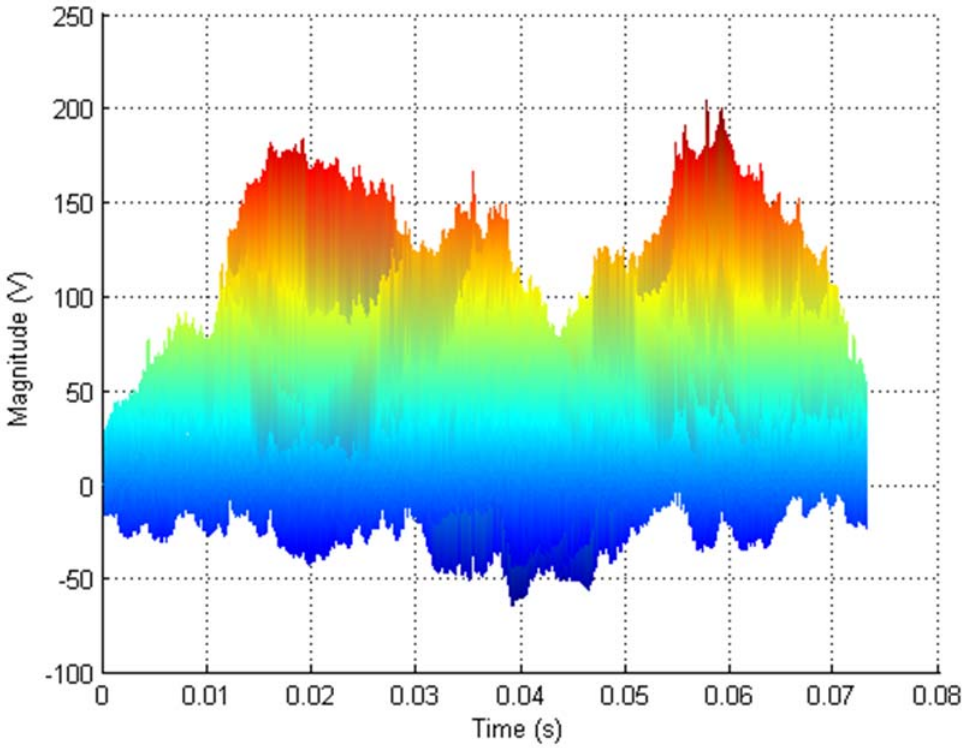

Figure 4: Marginal Time Distribution of the FMCW Test Signal Using MATLAB CWD Software.

The final perspective is a time-frequency plot of the modulation as shown in Figure 5. The carrier frequency, the modulation bandwidth and the modulation period can all be identified from this perspective. The signal is visible at the carrier frequency of 1 kHz. The modulation period, which was unclear in the previous figures, can now be seen.

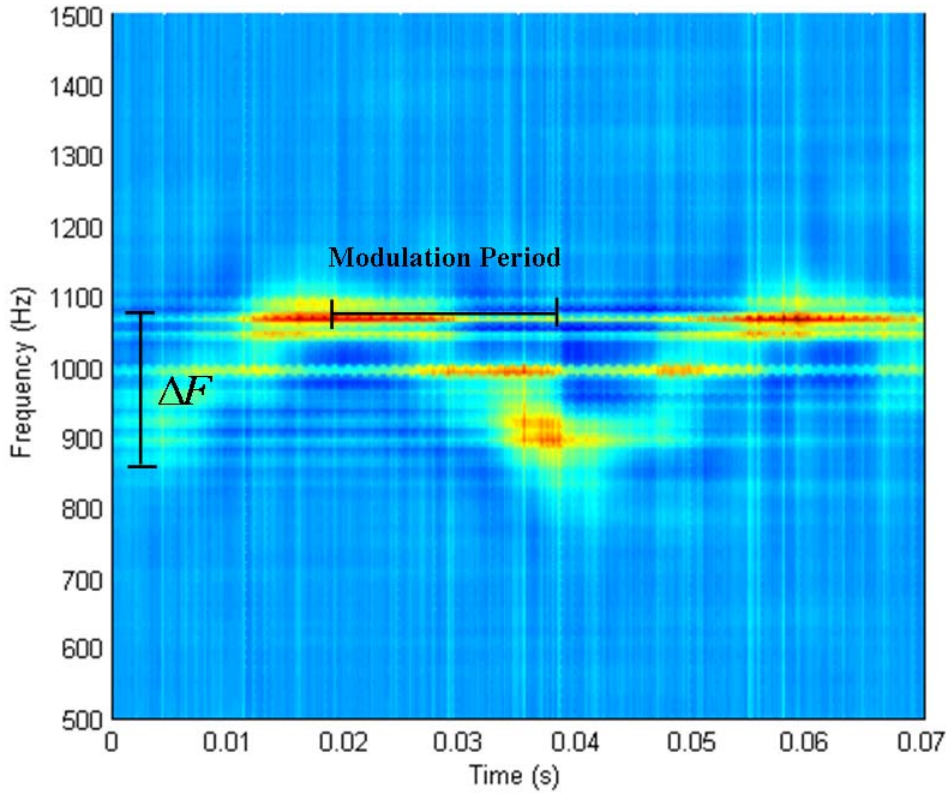

Figure 5: Time - Frequency Plot of an FMCW Test Signal Using MATLAB CWD Software.

Figures 2 thru 5 above are examples of the output of the CWD software code. Signal characteristics can be clearly identified and extracted from the graphs due to the reduction of cross-terms. Producing these graphs in a real-time environment and coupling it with an automatic identification algorithm would be advantageous in many military applications.

3. Choi-Williams Distribution C Benchmark

The MATLAB code was first converted into C code to be translated to the SRC-6. A step by step verification process of the C code was performed to verify the Choi-Williams distribution technique. After the code was verified on a small scale, the code was tested against a variety of fully developed signals generated by the LPI Toolbox.

B. 4 BY 4 INPUT SIGNAL VERIFICATION

For an N by N verification, a portion of a generated LPI signal was used. N must be a power of two therefore an LPI signal with a length of four samples was used. A signal of four samples is not long enough to determine signal characteristics but was sufficient to verify the software.

1. Input Signal

The input signal is described below in Table 1. For the code, the input signal must be a readable file with a column of real (I) data and a column of imaginary (Q) data.

Table 1. Input Signal Data for Code Verification.

I Data	Q Data
0.423889315973369	-0.576110684026631
1.323504414322150	1.481846094931446
-1.482367378829204	-0.284918532691066
-0.932103072187372	0.402749534832605

2. Wigner-Ville Distribution

The first step in carrying out the Choi-Williams distribution is to calculate the Wigner-Ville distribution and then modify that output with Choi-Williams weighting functions. In MATLAB, this step was a single for loop. In the resulting C code this step was a multiple stage loop shown in Appendix B. Each step had to be broken out into a separate loop and included in an overall loop.

The output of the MATLAB code (variable “WV”) is shown in Table 2. This data can be compared to the output of the C code (variables “IWV” and “QWV”) found in Table 3. Data in Tables 2 and 3 have been rounded to fit the table format. MATLAB and the C code uses 15 decimals of precision.

Table 2. MATLAB Code Output for Wigner-Ville Distribution.

	Column 1	Column 2	Column 3	Column 4
Row 1	1.0310	0	0	0
Row 2	2.2786	-0.6368 - 1.9143i	0	-0.6368 + 1.9143i
Row 3	3.9475	-0.4642 + 0.9748i	0	-0.4642 - 0.9748i
Row 4	0.5116	0	0	0

Table 3. C Code Output for Wigner-Ville Distribution.

	Real Data (I)			
	Column 1	Column 2	Column 3	Column 4
Row 1	1.0310	-0.0000	0.0000	-0.0000
Row 2	2.2786	-0.6368	0.0000	-0.6368
Row 3	3.9475	-0.4642	0.0000	-0.4642
Row 4	0.5116	0.0000	0.0000	0.0000
	Imaginary Data (Q)			
	Column 1	Column 2	Column 3	Column 4
Row 1	-0.0000	-0.0000	-0.0000	-0.0000
Row 2	-0.0000	-1.9143	-0.0000	1.9143
Row 3	-0.0000	0.9748	-0.0000	-0.9748
Row 4	-0.0000	-0.0000	-0.0000	-0.0000

Notice that the solutions to the Wigner-Ville distribution are the same for both sets of code. The only difference is the way in which the complex numbers are stored. The final output of the Wigner-Ville distribution is an N-by-N array.

3. Choi-Williams Kernel Function

Once the Wigner-Ville distribution is calculated the Choi-Williams kernel function, $S'(\ell, n)$, can be determined and applied to the Wigner-Ville output array. In MATLAB, using array functions, the kernel can be applied using a single nested for loop. The C code implementation had to be broken into several steps.

The output from the MATLAB code for the Choi-Williams distribution before the FFT can be found in Table 4 and is labeled in the MATLAB code “kern” while the output of the equivalent C code is shown in Table 5 and is labeled “Ikern” and “Qkern”.

Table 4. MATLAB Code Output for Choi-Williams Distribution.

	Column 1	Column 2	Column 3	Column 4
Row 1	0.5116	3.9475	2.2786	1.0310
Row 2	-0.1681 - 0.0155i	-0.2709 + 0.1456i	-0.2816 + 0.3259i	-0.1881 + 0.3194i
Row 3	0.0000	0.0000	0.0000	0.0000
Row 4	-0.1681 + 0.0155i	-0.2709 - 0.1456i	-0.2816 - 0.3259i	-0.1881 - 0.3194i

Table 5. C Code Output for Choi-Williams Distribution.

	Real Data (I)			
	Column 1	Column 2	Column 3	Column 4
Row 1	0.5116	3.9475	2.2786	1.0310
Row 2	-0.1681	-0.2709	-0.2816	-0.1881
Row 3	0.0000	0.0000	0.0000	0.0000
Row 4	-0.1681	-0.2709	-0.2816	-0.1881
	Imaginary Data (Q)			
	Column 1	Column 2	Column 3	Column 4
Row 1	-0.0000	-0.0000	-0.0000	-0.0000
Row 2	-0.0155	0.1456	0.3259	0.3194
Row 3	-0.0000	-0.0000	-0.0000	-0.0000
Row 4	0.0155	-0.1456	-0.3259	-0.3194

Again, the data from both source codes is comparable, only the way in which the complex numbers are stored varies from MATLAB to C code. This shows that the data manipulation prior to completing the FFT produces the same results.

4. Fast Fourier Transform Function

In MATLAB, the FFT is accomplished by using the built-in fft() code available. Neither the standard library, nor the math library in C, contains an equivalent function.

Therefore a separate FFT had to be generated. Code supplied by Professor Jerome R. Breitenbach of California Polytechnic State University was used to accomplish the FFT needed [11]. This code can be found in Appendix C.

The final result of the MATLAB code contains only the real portion of the FFT and can be found in Table 6. Table 7 contains the real portion of the equivalent C code output. From this final output there are differences in the fourth decimal place. This is most likely due to differences in the implementation of the FFT.

Table 6. MATLAB Code Output for a 4 by 4 Input Matrix.

	Column 1	Column 2	Column 3	Column 4
Row 1	0.3509	6.8116	3.4307	1.3097
Row 2	0.9612	8.4774	5.8606	3.3396
Row 3	1.6955	8.9785	5.6837	2.8144
Row 4	1.0852	7.3128	3.2538	0.7845

Table 7. C Code Output for a 4 by 4 Input Matrix.

	Column 1	Column 2	Column 3	Column 4
Row 1	0.3509	6.8116	3.4306	1.3097
Row 2	0.9612	8.4774	5.8606	3.3397
Row 3	1.6955	8.9785	5.6837	2.8144
Row 4	1.0852	7.3127	3.2538	0.7844

5. 4 by 4 Verification Results

From this detailed analysis it can be shown that the C code functions as designed at each critical step of the software code. With this knowledge it can now be shown that the C code will function as desired when larger input matrices are used. From a practical standpoint, a larger input matrix is equivalent to a larger sample size of an unidentified LPI signal.

C. 512 BY 512 INPUT SIGNAL VERIFICATION

Practically, it is difficult to show the same steps as shown in previously using tables. The identical graphing function in MATLAB will be used on both the matrix output of the MATLAB code and the C code to make an accurate comparison. The code can be found in Appendix D. Figure 5 showed the results of a MATLAB CWD calculation with N equal to 512. Figure 6 shows the output of the C CWD calculation.

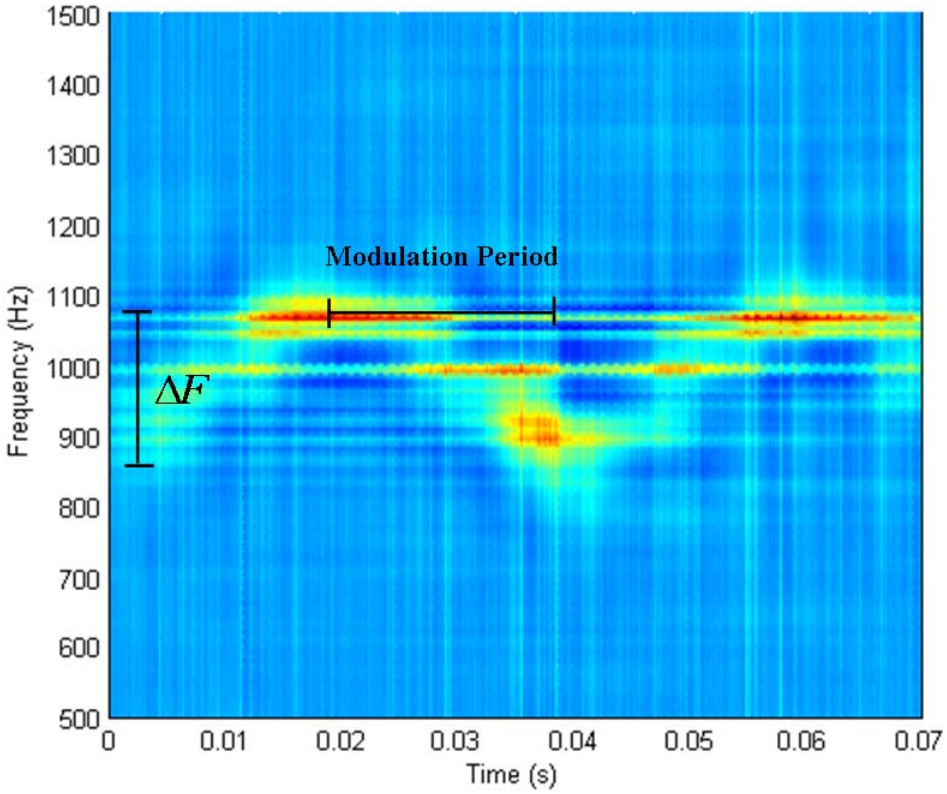

Figure 6: Time vs. Magnitude Plot FMCW test signal using C code CWD software.

There is very little visible difference between Figure 5 and Figure 6. A close inspection reveals minor differences that can be contributed to the different FFT algorithm. This is a final confirmation that the C code performs as expected when compared to the MATLAB output.

It should be noted that an N of less than 512 begins to decrease the number of signal characteristics that can be determined. A sample size of 256 would reduced the

above graph to half of the time scale. This would make the modulation period difficult to detect. These results may vary depending on what type of LPI signal is being analyzed.

D. SOFTWARE GENERATION – CONCLUSION

The CWD was a time-frequency analysis technique that used an exponential kernel, or weighting function, for signal processing. MATLAB code for the CWD was analyzed for a signal with a length of 512 samples. Standard C code was generated and a step by step verification was conducted with a signal length of four. These results were then expanded to verify that the C code functioned correctly for a signal length of 512 samples.

In the next chapter, the SRC-6 was reviewed and the standard C code was modified to utilize SRC-6 resources. Several signals were analyzed to verify that the algorithm was successfully modified from the original MATLAB.

III. SRC-6 IMPLEMENTATION

A. BACKGROUND

The final stage of converting the MATLAB code for use on the SRC-6 is to make some modifications to the C code developed above. The goal of converting the CWD technique to function on the SRC-6 is to increase the speed of the computation.

1. SRC-6 Hardware Background

The SRC-6 is a reconfigurable computer which can be tailored for a specific objective. Figure 7 shows a generic overview of the system architecture. There is a microprocessor board, a Multi-Adaptive Processing (MAP) board, and a common memory. The MAP is interfaced to the microprocessor board via Direct Memory Access (DMA) Procedures. Both the MAP and the microprocessor board have access to a common memory.

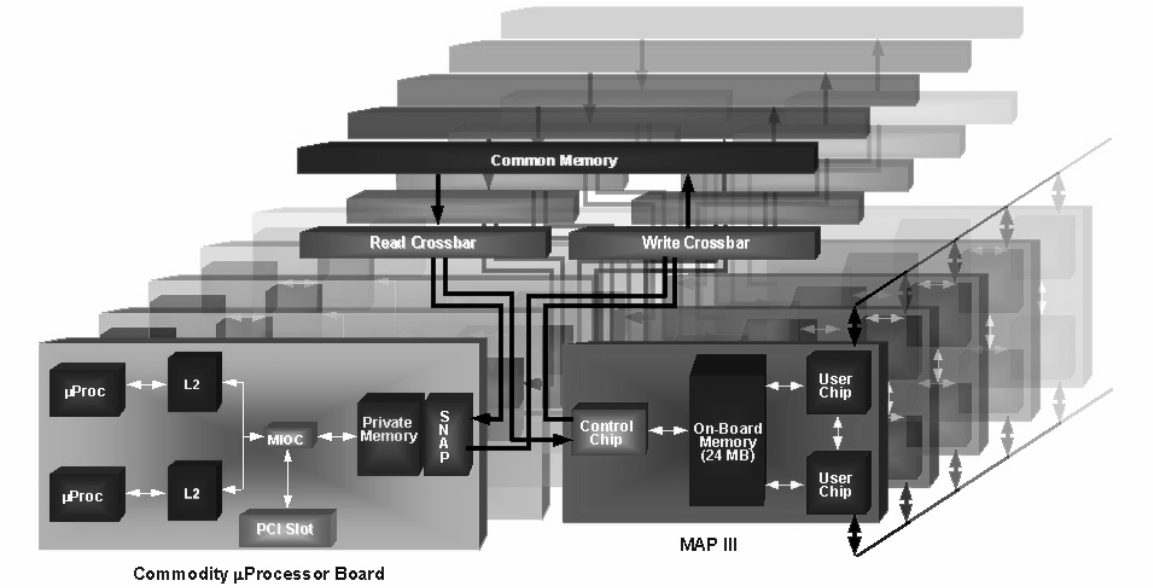

Figure 7: SRC-6 System Architecture. (From: [12])

A diagram of the SRC-6 at NPS can be found in Figure 8. The SRC-6 has two microprocessor boards. Each microprocessor board contains two Intel® Xeon™ 2.8 GHz processors. There are five Multi-Adaptive Processing (MAP) boards available, two MAP B boards and three MAP E boards. The B series board was not used for this thesis.

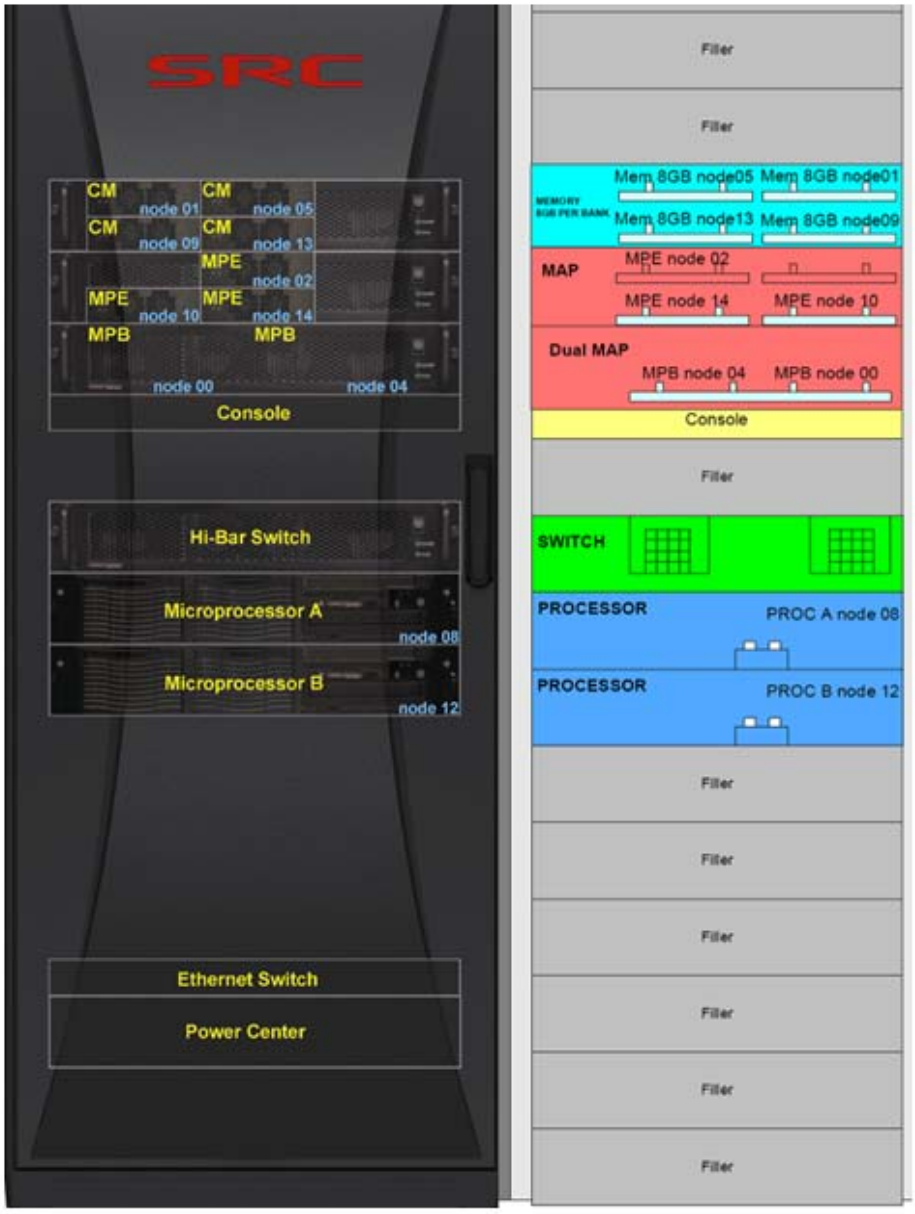

Figure 8: Naval Postgraduate School SRC-6 Diagram.

The MAP E board contains two Virtex-II Pro XC2VP100 Platform Field Programmable Gate Arrays (FPGAs) which perform the user logic sent to the MAP. There are seven on-board memory banks with a total bandwidth of 11.2 gigabytes/second. The MAP also has two general purpose input/output ports for MAP to MAP connections or other data input [13]. Figure 9 shows the outline of a MAP board.

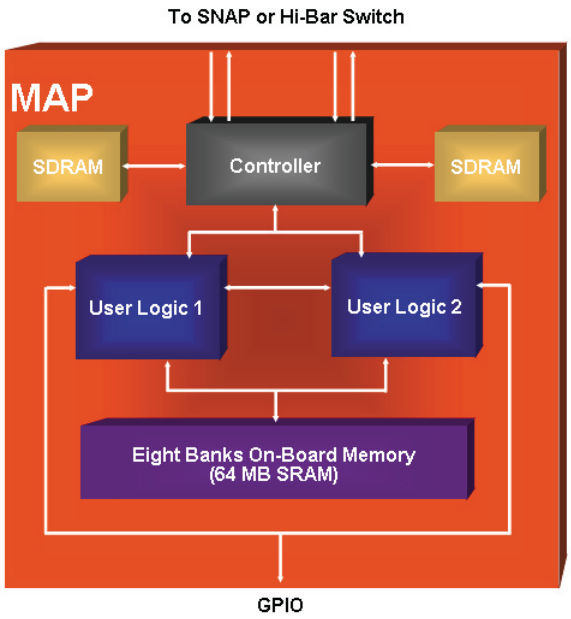

Figure 9: SRC-6 Multi-Adaptive Processing (MAP) Boards Diagram. (From: [13])

The MAP board is the key to the SRC-6's potential performance improvement over the standard MATLAB routine execution on a standard computer. Each MAP board is independent of the microprocessor and has access to the common memory. The MAP board is controlled by a command list that controls functions such as the DMA interactions. The FPGAs in the MAP board contain the user defined algorithm. Individual MAP boards can interact between other MAP boards without using the memory bandwidth in the system [14].

Each MAP can be used to perform repeated algorithms found in the overall larger code. For example, the FFT algorithm used in the MATLAB and C code can be run separately on the MAP unit and not access other common memory used for other functions in the code.

An LPI detection system benefits from running multiple algorithms simultaneously on the same data set. Algorithms can be tailored to maximize the performance of each detection strategy and the autonomous classification of LPI signals.

2. SRC-6 Software Configuration

Three types of files are needed to execute the algorithm generated in Chapter II.A.3 on the SRC-6 system. These files are the Makefile, the main file, and the MAP routines. Each type of file is outlined below. Refer to [14] for more information on each type of file and its construction.

The Makefile defines what preprocesses have to occur prior to compilation. It also includes instructions for the type of complier to be used, flags used for debugging, and floating point standards. The Makefile also specifies which main file, map routines, and user macros (if any) should be compiled. The name of the output file is also established. The Makefile is used to compile all the routines at once. A template is given in [14].

The second type of file is the main body of the algorithm, usually generated in C or Fortran. This main file may contain the brunt of the algorithm setup and definition. This portion of code is run on the microprocessor(s) of the SRC-6.

The largest computational function in the algorithm is defined in the third type of file, the MAP routine files. These are the files to be specifically run on the MAP processor. Several MAP routines can also be identified and use the same MAP board for different portions of the algorithm. No MAP routine has to be defined if all of the algorithm will be processed using the microprocessor. However, without utilizing the MAP board all the benefits of a reconfigurable computer are lost.

3. SRC-6 Summary

In summary, to utilize the benefits of the SRC-6, the C code generated in Chapter II must be modified to take advantage of the MAP boards. This modification allows more repetitious sections of code to be performed on the FPGAs on the MAP boards which can increases the overall speed of the computation. Due to the configurable nature of the

SRC-6 there are many ways to modify the original C code to take advantage of the MAP boards. One implementation is identified in the next section.

B. CODE CONVERSION

The code in Chapter II was modified to run on the SRC processors and MAP boards. Three files were generated to run the Choi-Williams Distribution. Each file is detailed below.

1. Main Code

The main code, named choiSRC.c, needed very little modifications. Overall, the code functions the same as it did previously. Most of the modification required was necessary to interface with the function located on the MAP board. The main code can be found in Appendix E.

First, to use the MAP board, the board itself had to be allocated and set aside for the function. This is accomplished using a `map_allocate(X)` command. Similarly, after the function has been called and the MAP board is no longer needed a `map_free(X)` command is used.

Second, the arrays that are sent to the function have to be aligned in memory. To facilitate this function the SRC has a built in function called `Cache_Aligned_Allocate(size)`. This function positions a pointer along a cache-aligned buffer and replaces all the `malloc` commands found in the original C code.

Last, the FFT algorithm used in the original C code was a recursive code that could not be directly translated to the MAP board. Because of this a different FFT algorithm was used. This algorithm required the generation of a twiddle table which was used as a weighting table in the FFT algorithm. This table was based on the length N of the input signal.

2. MC Code

The FFT algorithm was the function that was ported to the MAP board. The original C code FFT algorithm was not easily portable to the MAP board therefore a new FFT algorithm was needed. The SRC Computers Corporation provided NPS with an FFT library that could be used directly by the MAP board and sample code on how to efficiently use the library [15]. The sample code was modified to fit the specific needs of the original C code and increase the efficiency of the code for this specific use. A copy of this code can be found in Appendix F.

3. Makefile Code

The Makefile used for this code began with the standard template. The sections for `FILES`, `MAPFILES`, and `BIN` were appropriately filled out. To run the FFT provided by SRC Computers, extra lines to include the `FFT_LIB` were added. Also, the flags were set to `-log` so that a log file would be generated after the code was complied. The code can be found in Appendix G.

C. CODE VERIFICATION - FMCW

An N by N comparison where N is equal to 512 was conducted to verify the SRC C code is functioning as designed. First, the signal used in the previous section was used to verify the SRC C code results. Second, the code was run against three other LPI signals to confirm that the software is robust and can support a variety of LPI signals.

1. FMCW LPI Signals

The LPI signal used in Figures 5 and 6 was again be used for verification of the SRC C code. The signal used was an FMCW with a carrier frequency of 1 kHz, a sampling frequency of 7 kHz, a modulation bandwidth of 250 Hz, a modulation period of 20 μs and a SNR of 0 dB. Figure 10 shows the results of the SRC C code using MATLAB to graph the results.

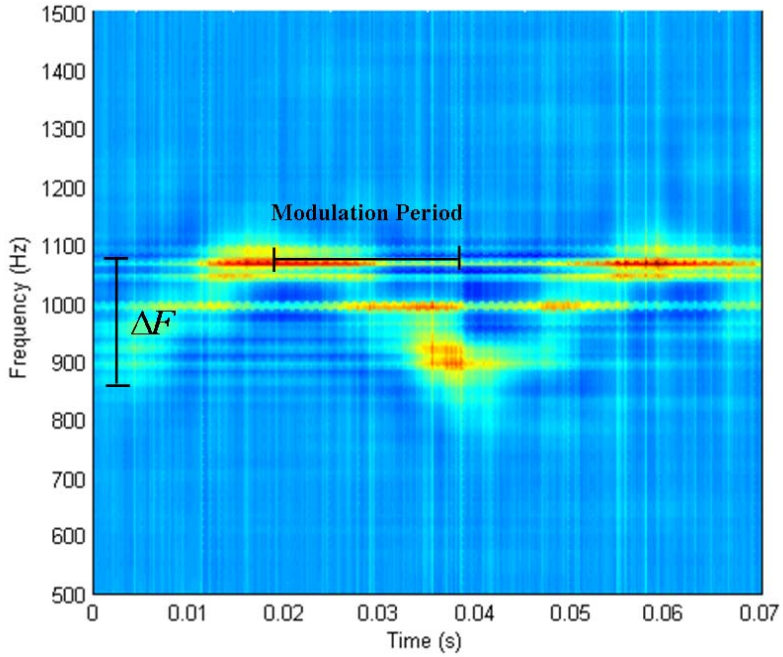

Figure 10: Time - Frequency Plot of an FMCW Test Signal Using SRC C Code CWD Software.

A comparison between Figures 5, 6 and 10 show that the output is indeed similar. It is important to note that these results do not show the magnitude of the output. Due to the different FFT algorithms used at each stage (MATLAB, C, and SRC) the magnitudes of these graphs are all different. For use in LPI detection and identification however, the magnitude of these plots is irrelevant as long as the ratios are maintained as shown in the graphs above.

2. Frank Code LPI Signals

Frank codes are polyphase codes that are derived from linear frequency modulated (FM) waveforms. This code approximates a linear FM using a finite number of frequency steps. The Frank code had a carrier frequency of 1 kHz with four frequency steps and one cycle per step. The period was 16 ms and the SNR was 0 dB. It was sampled at a rate of 7 kHz. For more information on Frank codes see [6], Section 5.6. Figure 11 shows a comparison between the MATLAB code and the SRC C code.

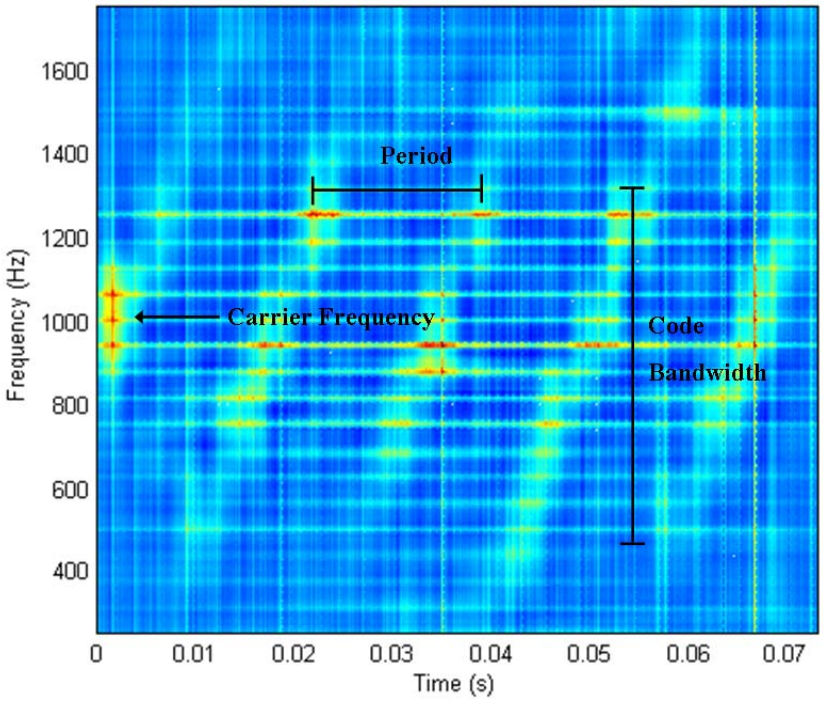

Figure 11a: MATLAB Results on a Frank Code LPI Signal.

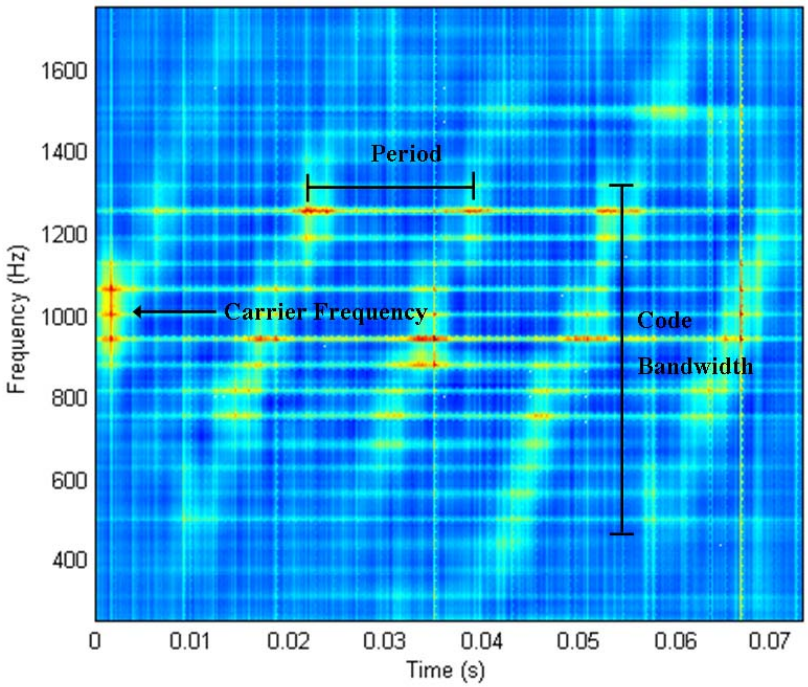

Figure 11b: SRC Results on a Frank Code LPI Signal.

There is no discernable difference between Figures 11a and 11b. The LPI signal is very visible against the 0 dB of noise. The code period of 16 ms is well defined as well as the code bandwidth. The carrier frequency is the middle point of the bandwidth and is 1 kHz.

3. Costas Code LPI Signals

Costas codes are sequences of frequencies that are best suited for unambiguous range and Doppler measurements with limited amount of interference between frequencies. The Costas codes used for this example are [3000, 2000, 6000, 4000, 5000, 1000] Hz. Each frequency is on for a duration of 0.005 seconds. The ADC has a sampling frequency of 15,057 Hz and the SNR is 0 dB. For more information on Costas codes see [6], Section 6.4. Figure 12 shows the results of both the MATLAB and the SRC implementations.

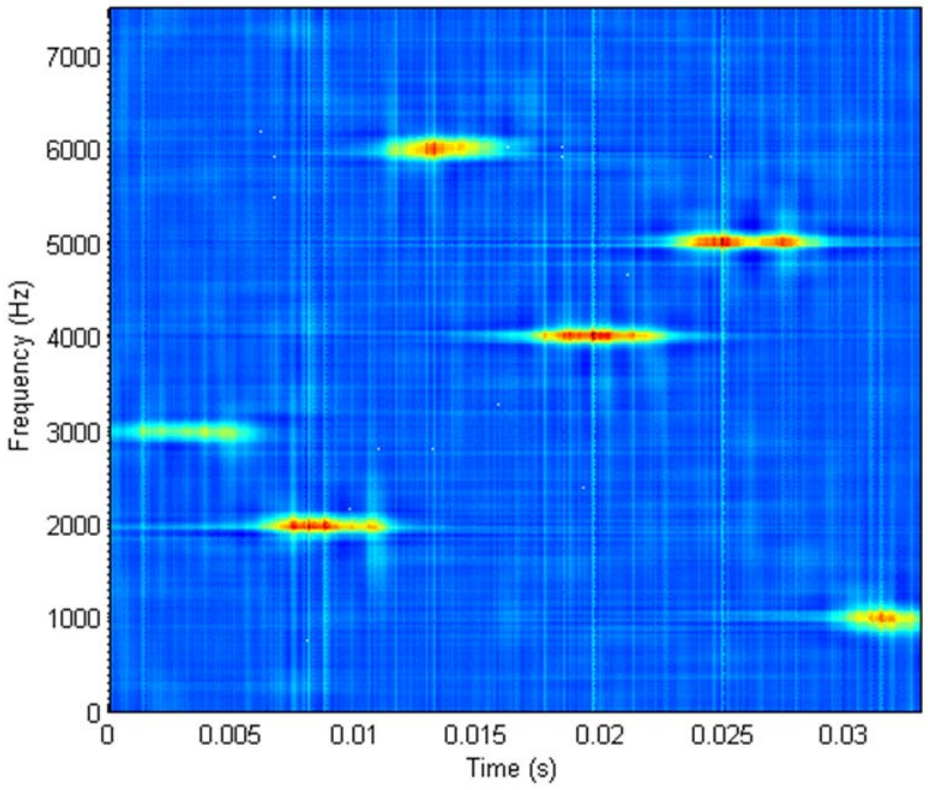

Figure 12a: MATLAB Results on a Costas Code LPI Signal.

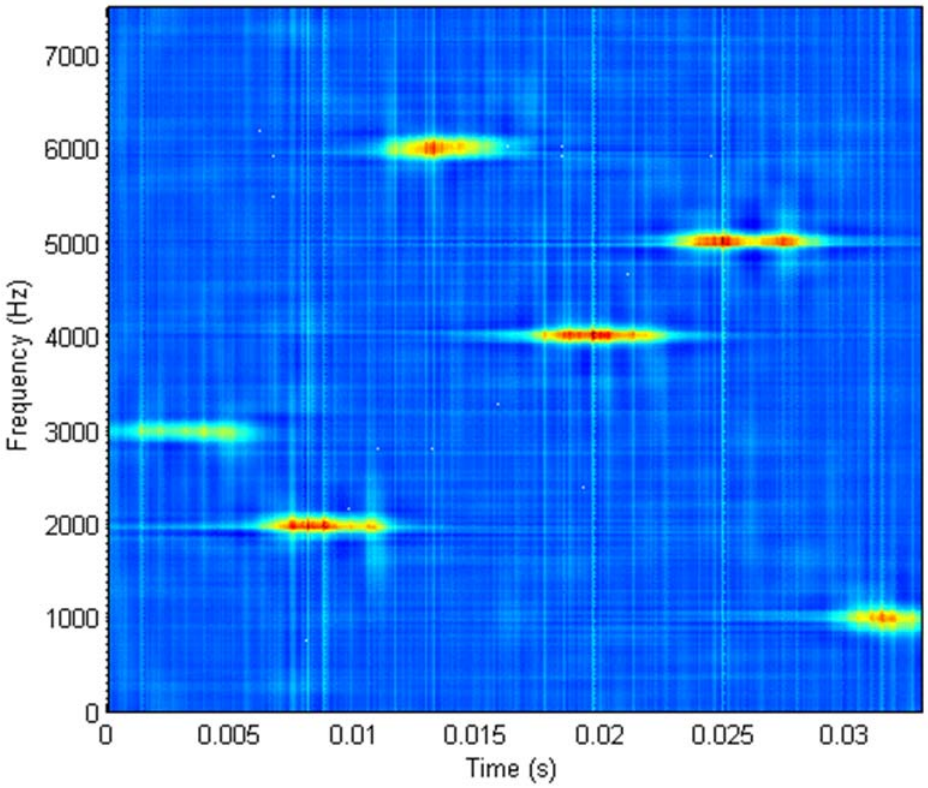

Figure 12b: SRC Results on a Costas Code LPI Signal.

Both Figure 12a and 12b show the same results using two different coded algorithms. Note that each frequency is noted as a different color in the graph and they occur along the time axis in the order in which they were originally transmitted. The Costas sequence used by the LPI signal is very apparent without any prior knowledge of the signal.

4. Hybrid Code LPI Signals

A hybrid LPI signal is a signal that combines characteristics from both frequency modulation techniques and phase modulation techniques. A 5-bit Barker code is used for each frequency in the Costas sequence and the Costas sequence is the same sequence used in Section C.3. For more information on hybrid LPI signals see [6], Section 6.5. The results are shown in Figure 13.

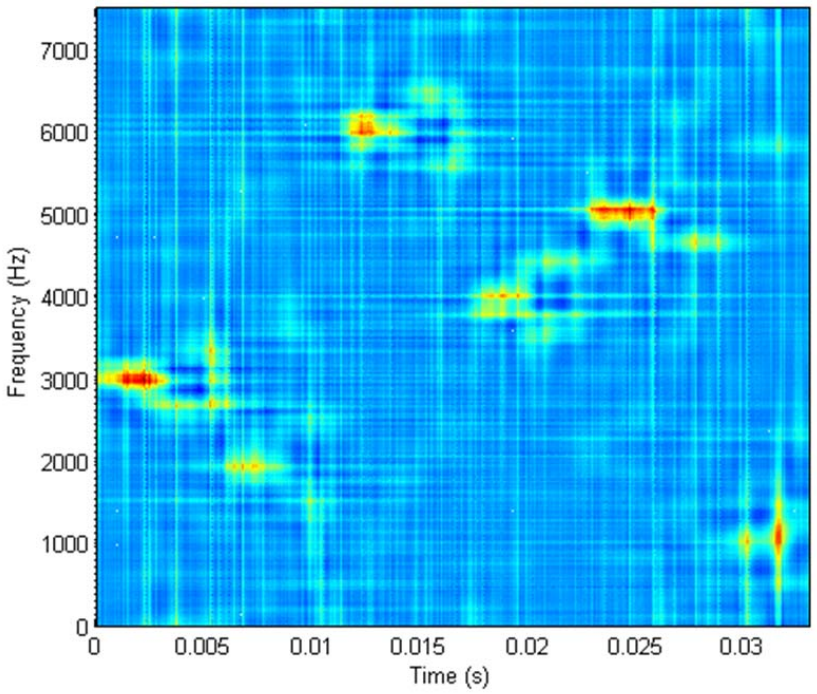

Figure 13a: MATLAB Results on a Hybrid Costas Code LPI Signal.

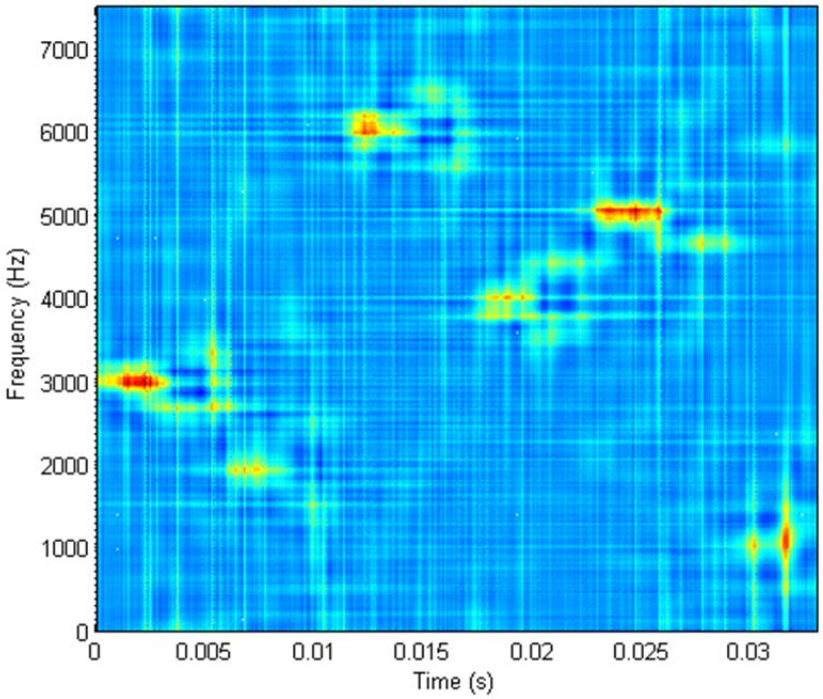

Figure 13b: SRC Results on a Hybrid Costas Code LPI Signal.

Figures 13a and 13b appear to be identical. Because of the dual LPI signal techniques used on this signal it is harder to identify the six Costas codes but they are visible in the diagram. The 5-bit Barker code is only visible by the spreading of the frequency “spikes”. Another detection strategy may be more practical for determining the Barker code sequence.

D. CODE VERIFICATION – CONCLUSION

From the analysis of the above four signals it can be concluded that the SRC implementation of the SRC code is functioning as designed and compares favorably to the results generated by MATLAB. An analysis of the algorithm’s speed on the three software platforms will be conducted in the next chapter.

IV. PERFORMANCE EVALUATION

A. BACKGROUND

The objective of this thesis was to investigate the feasibility of using the SRC-6 reconfigurable computer to compute the CWD on LPI signals. To this point the code has been verified and the CWD can be processed on the SRC-6. For a complete evaluation the final step is to determine the speed at which the CWD algorithm can be performed. For comparison purposes, the speed of the SRC algorithm utilizing the MAP boards will be compared against both the speed of the MATLAB code and the C code.

The MATLAB code was timed twice, once on a 3.0 GHz Pentium 4 Windows-based networked computer with 512 megabytes of random access memory using MATLAB 7.4.0 R2007b and again on a stand alone computer with a 1.6 GHz processor with one gigabyte of random access memory using MATLAB Student Version 7.1. The C and the SRC-6 C code were run on the Linux-based SRC-6 processor utilizing the same C compiler to help ensure a valid comparison.

B. ASSUMPTIONS

The following assumptions were used to make the most accurate comparison possible:

- All input LPI signals were a length of 512 samples long.
- The graphing functions were assumed to take the same amount of time and were not included in the overall timing of the algorithm. The results of the CWD would be input to the next phase of an automated LPI detection and classification system (such as Figure 1) and would not be graphed as output for a user to view.
- The time to read in a file or read out a file was not included. While this function was necessary for the SRC output, it was not generated in the MATLAB code. In an LPI detection system the input would be fed directly to the hardware and the output would be sent to the next phase, not converted to an output file.
- In MATLAB the `tic` and `toc` functions were used. The output of these functions is a total elapsed time in seconds.

- The C and SRC code was timed using the structure found in Appendix H. This structure allowed for more precise timing than the `time()` function. The output was in seconds. The hardware dependency of the timing function was reduced by using the SRC compiler for both SRC and the C code. All timing was accomplished when there were no other users utilizing any SRC resources.
- A total of 20 trials were conducted for each algorithm. These were then averaged to generate a mean value. The results of all the trials can be found in Appendix I.

C. RESULTS

The timing results can be found in Table 8 below.

Table 8. Time Results for Choi-Williams Distribution Code.

LPI Signal	File Name	File Size	MATLAB Code[1] (sec)	MATLAB Code[2] (sec)	C Code (sec)	SRC C Code (sec)
FMCW	F_1_7_250_20_0	58 kB	42.36	34.08	6.85	36.64
Frank Codes	FR_1_7_4_1_0	29 kB	42.38	34.26	6.85	36.56
Costas Codes	C_1_15_5000_0	133 kB	42.42	34.30	6.87	37.44
Hybrid Costas Codes	FSK_PSK_Costas_15_5_0	53 kB	42.44	34.31	6.86	36.96

Note: C code timing utilized the C compiler on the SRC but not the Multi-Adaptive Processing (MAP) board. The SRC Code timing used both the C compiler and the MAP board.

1: MATLAB Code was run on a 3 GHz processor with 512 megabytes of random access memory using MATLAB 7.4.0 R2007b.

2: MATLAB Code 2 was run on a 1.6 GHz processor with one gigabyte of random access memory using MATLAB Student Version 7.1.

From the above results there are several important items to note:

- It can be seen that the type of LPI signal did not have any affect on the overall timing regardless of the code used.
- The file size did not have any affect on the length of time to run the code. The CWD algorithm only used the first 512 samples, regardless of the number of samples in the file.
- The C code was the fastest code and the slowest code was the MATLAB code run on a 3 GHz processor with 512 megabytes of random access memory using MATLAB 7.4.0 R2007b.

1. MATLAB Results

An analysis of the MATLAB code shows that the majority of the time is spent computing the CWD kernel. This is the section of code that computes the exponential kernel of the CWD and gives the CWD an advantage over the Wigner-Ville distribution by reducing cross terms. There were no apparent areas in the loop that could be altered to increase the speed of the code.

The MATLAB code was timed on two different computers. Timing was dependent on the version of MATLAB and the hardware used. If it is assumed that the Student Version would not outperform the full version of the software then the speed increase must be due to either the availably of more random access memory or the absence of background processes typically present on a networked computer. Timing results of different aspects of the MATLAB code can be seen in Table 9. It can be seen that approximately 80% of the time was spent in the CWD kernel portion of the algorithm.

Table 9. Time Results for Choi-Williams Distribution (CWD) MATLAB Code.

LPI Signal	MATLAB Code[1] (sec)	CWD Kernel[1] (sec)	FFT[1] (sec)	MATLAB Code[2] (sec)	CWD Kernel[2] (sec)	FFT[2] (sec)
FMCW	42.36	33.56	0.03	34.08	27.40	0.02
Frank Codes	42.38	33.59	0.03	34.26	27.52	0.02
Costas Codes	42.42	33.63	0.03	34.30	27.56	0.02
Hybrid Costas Codes	42.44	33.66	0.03	34.31	27.52	0.02

1: MATLAB Code was run on a 3 GHz processor with 512 megabytes of random access memory using MATLAB 7.4.0 R2007b.

2: MATLAB Code 2 was run on a 1.6 GHz processor with one gigabyte of random access memory using MATLAB Student Version 7.1.

2. C Code Results

The C Code was also analyzed to determine if the speed could be increased. A straight conversion of the MATLAB code to C code, compiled using `gcc`, is very inefficient and took approximately 98 seconds to run. This code was then reviewed to determine if there were areas that could be improved on. Redundant computations were

removed and calculations were performed congruently within one loop versus individual loops for each task. This efficient use of loops and streamlining of the code removed 36 seconds from the overall speed and was approximately 36 seconds.

Next the compiler itself was reviewed. Using `icc` in place of `gcc` improved the speed of the code. Then the compiler options were reviewed. The code streamlined previously to approximately 36 seconds was reduced to approximately 6.5 seconds using the compiler options: `-O3 -tpp7 -xW -align -Zp16 -ipo –static`. "`-O3`" is an optimization option that vectorizes loops in the code to perform more efficiently. "`-tpp7` and `–xW`" are optimization codes specifically designed to work with Pentium 4 processors. The remaining options affect the way memory is used in the processor. For more information on the options used see [16].

Table 10 shows the results of timing the C code. It can be seen that the majority of the C code time, approximately 97%, was spent computing the CWD kernel. Execution time of the FFT was longer in the C code than in MATLAB and the SRC code.

Table 10. Time Results for Choi-Williams Distribution (CWD) C Code.

LPI Signal	C Code (sec)	CWD Kernel (sec)	FFT (sec)
FMCW	6.85	6.66	0.08
Frank Codes	6.85	6.66	0.08
Costas Codes	6.87	6.68	0.08
Hybrid Costas Codes	6.86	6.67	0.08

3. SRC C Code Results

The optimized C code was used in developing the SRC C code using the MAP resources. The compiler options used were: `-O3 -tpp7 -xW –ip`. The options to affect memory would not function on the SRC. However all pointers were instantiated using `Cache_Aligned_Allocate` verses `malloc` to align pointers at a cache boundary. Table 11 shows timing results of individual aspects of the SRC code.

Table 11. Time Results for Choi-Williams Distribution SRC C Code.

LPI Signal	Microprocessor (sec)	MAP (sec)	DMA (sec)	FFT (sec)	Call (sec)
FMCW	36.407	0.028	0.023	0.004	0.209
Frank Codes	36.329	0.027	0.023	0.004	0.206
Costas Codes	37.203	0.027	0.022	0.004	0.206
Hybrid Costas Codes	36.725	0.027	0.022	0.004	0.205

The Microprocessor time is average time the algorithm spent on the microprocessor. The MAP time was the average time it took to execute the entire MAP subroutine which consists of the DMA time and the FFT time. The DMA time is the average amount of time it took to transfer data to the MAP. The FFT time was the average amount of time to complete the MAP routine. The Call Time is the average amount of time to transition from the main C code to the MAP routine. These values are the cumulative results throughout the entire program executed using 512 data samples. For example, the FFT algorithm took 0.004 divided by 512, or 0.000008 seconds, each time the algorithm was called. Also note, that the summation of the Microprocessor time, the MAP time, and the Call time are equal to the data shown in Table 8.

4. Results Summary

Each code implementation was slowest when executing the CWD kernel function. This function is integral to the distribution and cannot be removed without altering the distribution itself. However it may be able to be manipulated to achieve a faster program.

Each program implemented the FFT algorithm differently. The MATLAB code used the built-in `fft()` function. The C code utilized the code from Professor Jerome R. Breitenbach found in Appendix C. The SRC implementation used the FFT provided by SRC Computer, Inc. Table 12 shows the timing results of each FFT algorithm. From these results it can be seen that the SRC implementation was an order of magnitude faster than either the MATLAB or the C FFT implementation. It can also be seen that the timing is not dependent on the type of LPI signal.

Table 12. Time Results for the Fast Fourier Transform (FFT).

LPI Signal	MATLAB FFT[1] (sec)	MATLAB FFT[2] (sec)	C FFT (sec)	SRC FFT (sec)
FMCW	0.03	0.02	0.08	0.004
Frank Codes	0.03	0.02	0.08	0.004
Costas Codes	0.03	0.02	0.08	0.004
Hybrid Costas Codes	0.03	0.02	0.08	0.004

1: MATLAB Code was run on a 3 GHz processor with 512 megabytes of random access memory using MATLAB 7.4.0 R2007b.

2: MATLAB Code 2 was run on a 1.6 GHz processor with one gigabyte of random access memory using MATLAB Student Version 7.1.

It is important to note that timing the code itself added delays into the code at every step. The intrusion was minimal in the MATLAB code using the `tic` and `toc` functions. However in the C and SRC code an additional library was added, an additional .c file was included and multiple `gettimeofday()` commands and other timing commands were added to the code. These additions and extra calculations must add some delay to both the C code and the SRC code. The final results and conclusions are found in the next chapter.

V. RESULTS AND CONCLUSIONS

A. RESULTS

The timing results in the previous chapter show several important overall results. These results are summarized below.

MATLAB Code:

- The MATLAB code performed the slowest when utilizing 3 GHz processor with 512 megabytes of random access memory using MATLAB 7.4.0 R2007b.
- The MATLAB results were consistent regardless of what signal was processed.
- The MATLAB results were dependent on the specific computer being used and what other programs were running in the background. The speed was impacted by more than several seconds when the computer was processing several programs. The results are also dependent on the amount of random access memory available.

C Code:

- The C code performed the fastest.
- The C code results were consistent regardless of what signal was processed.
- The C code could be further optimized past the original line-by-line translation of the MATLAB code.
- Additional speed could be achieved using compilation options that were not available in MATLAB.

SRC C Code:

- The SRC C code was approximately six times (or 80%) slower than the original C code. The SRC results were faster than the MATLAB code using a 3 GHz processor with 512 megabytes of random access memory using MATLAB 7.4.0 R2007b. The SRC results were comparable to the MATLAB code using a 1.6 GHz processor with one gigabyte of random access memory using MATLAB Student Version 7.1.
- The SRC C code results were consistent regardless of what signal was processed.
- The SRC C code could not use all the compilation options that were used when the C code (without using the MAP board) was compiled.

- The C and SRC algorithms were compared with the respective FFTs removed. When the same compiler options were used, the C code took approximately 7 seconds where the SRC C code took approximately 36 seconds. This comparison utilizes the same code, the same processor, the same compiler, and the same compiler options. The difference in timing could not be resolved.

B. CONCLUSIONS

The Choi-Williams distribution uses a recursive exponential kernel function to remove cross terms in a time-frequency analysis of a complex signal. While the use of the kernel reduced cross terms and allows for an improved estimations of signal parameters, it is also computationally expensive.

The MATLAB implementation of the code was a straight-forward manipulation of the Choi-Williams distribution. This allows for an easier initial code generation. However, the extra tools available in MATLAB which made the coding simpler (complex numbers, matrix manipulation, etc.) generated overhead that cost the implementation speed.

The C code implementation was optimized and used compilation options to make loop calculations more efficient without affecting the function of the loop itself. This allowed the C code to be highly efficient and to perform the fastest.

The SRC C code utilizing the MAP board was extremely efficient in areas of the code that were implemented on the MAP itself but lost optimization on the sections of code that were still implemented on the main processor. There are several options to utilize the speed of the MAP:

- Take additional sections of the main code, like the Choi-Williams kernel, and process it on the MAP as a separate MC file. This would take a large portion of the work of the program and parse it out to the faster MAP board. However, a direct porting of the CWD kernel to the MAP would not be efficient due to the number of computationally difficult tasks such as division, square roots, and exponential function that occur in the CWD kernel.

- Investigate alternative coding schemes for the Choi-Williams kernel in the main program to see if it can be streamlined.
- Investigate alternative memory allocations for the algorithms. Data may be more efficiently managed using shared memory between the MAP and the microprocessor verses the common memory or the on board memory of the MAP.
- Attempt to use the Wigner-Ville distribution which has a kernel of one. This would eliminate the need for the exponential kernel but increase cross terms in the final results.

It is also important to note that the SRC can be programmed in a hardware language such as VHDL or Verilog. If the CWD algorithm was defined using either of these methods the inefficiencies caused by the C compiler would be removed. However, this is no guarantee that the program would perform faster.

The detection strategy used here is one of three detection strategies that are currently designed to run concurrently for real-time processing. The SRC-6 can be tailored to process all three channels simultaneously. The MATLAB software does not have this feature. Running all three detection strategies on the same computer will have a large impact on the overall timing of the system.

C. RECOMMENDATIONS FOR FUTURE WORK

The project of using the SRC-6 as an autonomous LPI detection system should be continued in future theses. Specifically the following areas should be addressed:

- Investigate alternatives in implementing the Choi-Williams algorithm defined in this thesis. Review loop reduction techniques and hardware language implementation. Introduce windowing functions as defined in [1].
- Optimize the use of parallel detection strategies on the SRC-6. With three possible detection strategies there is a high probability that there will be ways to combine MAP board functions to optimize MAP board usage. For example

both the Choi-Williams distribution and the cyclostationary analysis method use an FFT function. This function could be used by both detection strategies.

- The current preprocessing phase of image analysis has only been configured and tested for the quadrature mirror filtering detection strategy. This phase must be expanded to include the Choi-Williams Distribution.
- Review the use of the Wigner-Ville distribution in conjunction with the Choi-Williams distribution to maximize both the speed and the quality of the results. For a military application such as an radar warning receivers, a less than optimal result may be sufficient if it means the pilot has more warning time (i.e. faster results). For an ELINT application it may be more advantageous to use the Choi-Williams distribution to recover more signal parameters.
- The classification algorithms must be expanded to include signal characteristics identified by the Choi-Williams Distribution.

APPENDIX A. LPI SIGNAL GENERATION

There were several signals generated by the LPI Toolbox used in this thesis. Each signal is described in detail below. Each signal was generated with a signal to noise ratio of 0 dB. The addition of noise generates more realistic results without overwhelming the graph.

1) F_1_7_250_20_0.mat: This signal is an FMCW signal with a carrier frequency of 1 kHz and a modulation frequency (bandwidth) of 250 Hz. It has a modulation period of 20 μs and is sampled at 7 kHz.

2) FR_1_7_4_1_0.mat: This signal is a Frank coded polyphase signal that is related to frequency modulation and Barker codes. This code has a frequency of 1 kHz and has four frequency steps. The four steps leads to a period of 16 ms (four squared). There is one cycle per step and it is sampled at a rate of 7 kHz.

3) C_1_15_5000_0.mat: This signal is comprised of Costas codes that limit the amount of interference between frequencies and generate an ideal unambiguous range and Doppler measurement. The Costas code used was [3000, 2000, 6000, 4000, 5000, 1000] Hz. Each frequency is on for a duration of 0.005 seconds and a sampling frequency of 15,057 Hz.

4) FSK_PSK_Costas_15_5_0.mat: This signal is a combination of Frequency Shift Keying and Phase Shift Keying techniques. This example used a 5-bit Barker code over the range of each frequency in the Costas sequence [3000, 2000, 6000, 4000, 5000, 1000] Hz.

More information on the use of the LPI Toolbox and the different LPI signals is given in [6].

To use these signals in the SRC environment, they were converted to text files using the following code.

```
%% %%%%%%%%%%%%%%%%%%%%% FMCW Code %%%%%%%%%%%%%%%%%%%% %%
load H:\Thesis\Choi\Test_signals\F_1_7_250_20_0.mat
sig1 = [I Q];
save S:\thesis\Test_signals_txt\F_1_7_250_20_0.txt sig1 -ascii -double

%% %%%%%%%%%%%%%%%%%%%%% Frank Code %%%%%%%%%%%%%%%%%%%% %%
load H:\Thesis\Choi\Test_signals\FR_1_7_4_1_0.mat
sig2 = [I Q];
save S:\thesis\Test_signals_txt\FR_1_7_4_1_0.txt sig2 -ascii -double

%% %%%%%%%%%%%%%%%%%%%%% Costas Code %%%%%%%%%%%%%%%%%%%% %%
load H:\Thesis\Choi\Test_signals\C_1_15_5000_0.mat
sig3 = [I Q];
save S:\thesis\Test_signals_txt\C_1_15_5000_0.txt sig3 -ascii -double

%% %%%%%%%%%%%%%%%% FSK/PSK Costas Code %%%%%%%%%%%%%%%%% %%
load H:\Thesis\Choi\Test_signals\FSK_PSK_Costas_15_5_0.mat
sig4 = [I Q];
save S:\thesis\Test_signals_txt\FSK_PSK_Costas_15_5_0.txt sig4 -ascii
        -double
```

APPENDIX B. CHOI-WILLIAMS DISTRIBUTION C CODE

A file was generated to contain all the user defined variables. In the case of the C code there is only one variable. However to change the length of the vector being calculated only the user file needs to be altered.

```
/************** userdef.c *************************/
#define PI 3.14159265358979323846264338327795

#define N  512

//#define N  256
//#define N  8
```

```
#include <stdio.h>
#include <math.h>
#include <time.h>
#include "fft.c"
#include "userdef.c"
#include "high_prec_time.c"

/******* Choi Williams Distribution Computation ********
*******************************************************
* N = length of vector (file) Must be an Integer power *
*     of two (256 or 512 is recommended).              *
*                                                      *
* *data = pointer to input data                        *
* Data must be I and Q data in two respective columns  *
*                                                      *
* Output vector will be NxN                            *
* Rows represent change in time                        *
* Columns represent change in frequency                *
*******************************************************/

int main()
{
 /* Define Variables */
 int i,j,k,r,q,P,L,M,n,a,b;

 struct timeval tottime, stime, etime;
 struct timeval totFFTtime, sFFTtime, eFFTtime;
 struct timeval totCWDtime, sCWDtime, eCWDtime;

 double sigma,tt,W,c,ffttime,time1,time2;
 float  Iin[N], Qin[N];

 /* Define Pointers */
 FILE *data;

 double (*temp1)[2],      (*temp2)[2];
```

```
 double (*IData),          (*QData);
 double (*Ishift),         (*Qshift);
 double (*IX1),            (*QX1);
 double (*IX2),            (*QX2);
 double (*Isum),           (*Qsum);
 double (*IWV)[N],         (*QWV)[N];
 double (*IWVweight)[N-1], (*QWVweight)[N-1];
 double (*Ikern)[N],       (*Qkern)[N];
 double (*IDataOut)[N],    (*QDataOut)[N];
 double (*mu),             (*weight)[N];

/*****************Initialize Variables*****************
******************************************************/
 k = 1;
 j = 0;
 P = N/2;
 M= (2*P)-1;
 a = -P;
 b = P-1;
 ffttime = 0;

/******************** Sigma ***************************
******************************************************
Can use to change the weight of the Choi Williams
distribution. A large sigma will result in a
Wigner-Ville Distribution.                            */

sigma = 1;

/***************** Allocate memory *********************/
     IData = malloc(N*sizeof(double));
     QData = malloc(N*sizeof(double));

/****************** Open Data ************************
******************************************************
Default: Data is stored in a file called
              Test_signals_txt                        */

 data  = fopen("Test_signals_txt/INPUT.txt","r");

 if(data == NULL)
  {
   puts("Error opening file.");
   return(1);
  }

  fscanf(data,"%f", &Iin[0]);
  IData[0] = Iin[0];

  for(i=0; i<((2*N)-1);i++)
   {
    if(i % 2 == 1)                        /*IData*/
     {
      fscanf(data, "%f", &Iin[k]);
      IData[k] = Iin[k];
```

```
      k++;
     }
    else                                    /*QData*/
     {
      fscanf(data,"%f", &Qin[j]);
      QData[j] = Qin[j];
      j++;
     }
   }
/* Close the file */
fclose(data);

/************** Start Time the SRC Code ***************
 The total time will be from this point to the stage  *
 right before the output to txt. This is to make as   *
 fair a comparison between the SRC code, the C code   *
 and the Matlab code.                                 */

 gettimeofday(&stime,NULL);

/***** Compute the Wigner-Ville Distribution ***********
*****************************************************
The Wigner-Ville Distribution can be used to
computer the Choi-Williams Distribution. The
variables IWV and QWV are the NxN matrix output.

First, the data is shifted.
Second, the data is broken into X1 and X2.
Lastly, the data is recombined into IWV and QWV.       */

    /*************** Allocate memory ******************/
     Ishift = malloc(N*sizeof(double));
     Qshift = malloc(N*sizeof(double));
     IX1    = malloc(N*sizeof(double));
     QX1    = malloc(N*sizeof(double));
     IX2    = malloc(N*sizeof(double));
     QX2    = malloc(N*sizeof(double));
     IWV    = malloc(N*N*sizeof(double));
     QWV    = malloc(N*N*sizeof(double));
     Ikern  = malloc(N*N*sizeof(double));
     Qkern  = malloc(N*N*sizeof(double));

for(i=a; i<b+1; i++)
 {
/******************** Shift ***************************
 Shift both I and Q                                   */

  if(i>= 0)
   {
    for(k=0; k<i; k++)
     {
      Ishift[k] = 0;
      Qshift[k] = 0;
     }
    for (k=i; k<N; k++)
```

```
     {
      Ishift[k] = IData[k-i];
      Qshift[k] = QData[k-i];
     }
   }
 else
   {
    for(k=0; k<N+i; k++)
     {
      Ishift[k] = IData[k-i];
      Qshift[k] = QData[k-i];
     }
    for(k=N+i; k<N; k++)
     {
      Ishift[k] = 0;
      Qshift[k] = 0;
     }
   }
/**************** End Shift ***************************/
/************ Determine X1 and X2 **********************/
 for(k=0; k<P; k++)
  {
   IX1[k] = Ishift[P+k-1];
   QX1[k] = Qshift[P+k-1];
   IX2[k] = Ishift[P-1-k];
   QX2[k] = -Qshift[P-1-k];
  }

 IX1[k] = 0;
 QX1[k] = 0;
 IX2[k] = 0;
 QX2[k] = 0;

 for(k=(P+1); k<N; k++)
  {
   IX1[k] = Ishift[k-P-1];
   QX1[k] = Qshift[k-P-1];
   IX2[k] = Ishift[M+(M-P+1)-k];
   QX2[k] = -Qshift[M+(M-P+1)-k];
  }

/****************** End X1 and X2 **********************/
/******************** Determine WV **********************/
 for (k=0; k<N; k++)
  {
   IWV[i+P][k] = ((IX1[k]*IX2[k])-(QX1[k]*QX2[k]));
   QWV[i+P][k] = -((IX1[k]*QX2[k])+(IX2[k]*QX1[k]));
  }
/********************* End WV ***************************/
 }

    /*************** Release memory ********************/
     free(IX2);
     free(QX2);
     free(IX1);
```

```
     free(QX1);
     free(Ishift);
     free(Qshift);

/********** End Wigner-Ville Distribution **************
*****************************************************/
 gettimeofday(&sCWDtime,NULL);
/****** Compute the Choi-Williams Distribution *********
*****************************************************
The Choi-Williams kernel is a summation of weighted WV
outputs.                                              */

/****************** Determine mu **********************
 Mu is used for array indexing in determining the
 weighting function.                                  */

    /*************** Allocate memory ******************/
     mu = malloc(N*sizeof(double));

for(k=0; k<N; k++)
 mu[k]=b-k;

/******************** End mu ***************************/
/************* Determine weight ************************
 These nested for loops determine the weighting
 function in the Choi-Williams Distribution.          */

    /*************** Allocate memory ******************/
     weight    = malloc(N*N*sizeof(double));
     Isum      = malloc(N*sizeof(double));
     Qsum      = malloc(N*sizeof(double));
     IWVweight = malloc(N*N*sizeof(double));
     QWVweight = malloc(N*N*sizeof(double));

 c = sigma/(4*PI);

for(L=0; L<N; L++)
 {
  for(n=1; n<P; n++)
   {
    for(i=0; i<N; i=i+4)
     {
      W = sqrt(c/(n*n))
          *exp((-((mu[i]-(L-P))*(mu[i]-(L-P)))*sigma/(4*(n*n))));
      weight[i][n] = W;
      weight[i][N-n] = W;


      W = sqrt(c/(n*n))
          *exp((-((mu[i+1]-(L-P))*(mu[i+1]-(L-P)))*sigma/(4*(n*n))));
      weight[i+1][n] = W;
      weight[i+1][N-n] = W;


      W = sqrt(c/(n*n))
```

```
          *exp((-((mu[i+2]-(L-P))*(mu[i+2]-(L-P)))*sigma/(4*(n*n))));
      weight[i+2][n] = W;
      weight[i+2][N-n] = W;


      W = sqrt(c/(n*n))
          *exp((-((mu[i+3]-(L-P))*(mu[i+3]-(L-P)))*sigma/(4*(n*n))));
      weight[i+3][n] = W;
      weight[i+3][N-n] = W;

     } //End i loop
   }//End n loop

  for(j=0; j<N-1; j++) //columns
   {
    Isum[j] = 0;
    Qsum[j] = 0;
    for(k=0; k<N; k++) //rows
     {
      IWVweight[k][j] = weight[k][j+1] * IWV[k][j+1];
      QWVweight[k][j] = weight[k][j+1] * QWV[k][j+1];
      Isum[j] = Isum[j] + IWVweight[k][j];
      Qsum[j] = Qsum[j] + QWVweight[k][j];
     } //End k loop
    } //End j loop

/****************** End weight ************************/
/**************** Determine kernel *******************/

   for(r=0; r<N; r++)
    {
     //printf("L = %i; r = %i; N=%i\n",L,r,N);
     if(r==0)
       {
        Ikern[r][L] = IWV[N-L-1][r];
        Qkern[r][L] = 0;
       }
     else
       {
        Ikern[r][L] = Isum[r-1];
        Qkern[r][L] = -Qsum[r-1];
       }
     }

 } //End L loop

    /*************** Release memory ******************/
     free(IWVweight);
     free(QWVweight);
     free(Isum);
     free(Qsum);
     free(weight);
     free(mu);
     free(IWV);
     free(QWV);
```

```
/********** End Choi-Williams Distribution *************/
 gettimeofday(&eCWDtime,NULL);
/***************** Determine FFT ***********************
*****************************************************
The FFT algorithm is for an Nx1 array so it used in a
loop to calculate the FFT for an NxN array.           */

    /*************** Allocate memory ******************
     This is to determine the FFT.                    */
     temp1    = malloc(2*N*sizeof(double));
     temp2    = malloc(2*N*sizeof(double));
     IDataOut = malloc(N*N*sizeof(double));
     QDataOut = malloc(N*N*sizeof(double));

for (j=0; j<N; j++)
 {
  for (i=0; i<N; i++)
   {
    temp1[i][0] = Ikern[i][j];
    temp1[i][1] = Qkern[i][j];
   }
 gettimeofday(&sFFTtime,NULL);
  fft(N,temp1,temp2);
 gettimeofday(&eFFTtime,NULL);

   tt = timeval_subtract(&totFFTtime, &eFFTtime, &sFFTtime);
   time2 = totFFTtime.tv_sec + totFFTtime.tv_usec*1e-6;
   time1 = time2+ffttime;
   ffttime = time1;

  for (i=0; i<N; i++)
   {
    IDataOut[i][j] = 2*temp2[i][0];
    QDataOut[i][j] = 2*temp2[i][1];
   }
 } // End j loop

    /*************** Release memory ******************/
     free(temp1);
     free(temp2);
     free(Ikern);
     free(Qkern);

/********************* End FFT ************************/
/*************** End Time the C Code *****************
 The total time will be from this point to the stage   *
 right before the output to txt. This is to make as    *
 fair a comparison between the SRC code, the C code    *
 and the Matlab code.                                  */

 gettimeofday(&etime,NULL);
 tt = timeval_subtract(&tottime, &etime, &stime);
printf("Total time was %4.6f sec\n",
        tottime.tv_sec + tottime.tv_usec/1000000.0);
```

```
 tt = timeval_subtract(&totCWDtime, &eCWDtime, &sCWDtime);
printf("Total CWD kernel time was %4.6f sec\n",
        totCWDtime.tv_sec + totCWDtime.tv_usec/1000000.0);

printf("Total FFT time was %4.6f sec\n",ffttime);

/*************** Output Data ************************
*****************************************************
This outputs the final result to a text file called
out.txt that can be pulled into Matlab and graphed.    */
 data  = fopen("out.txt","w");

if(data == NULL)
 {
  puts("Error creating file.");
  return(1);
 }

for (r=0; r<N; r++)
 {
  for (q=0; q<N-1; q++)
    fprintf(data, "%12f\t", IDataOut[r][q]);
    fprintf(data, "%12f\n", IDataOut[r][q]);
 }

/* Close the file */
fclose(data);

puts("Data saved to out.txt");


/***************** End Output ************************
*****************************************************/

/***************** Release memory *********************/
free(IData);
free(QData);
free(IDataOut);
free(QDataOut);

return(0);
} // End main
```

APPENDIX C. FFT.C C CODE

```
/*-------------------------------------------------------------------------
   fft.c - fast Fourier transform and its inverse (both recursively)
   Copyright (C) 2004, Jerome R. Breitenbach.  All rights reserved.

   The author gives permission to anyone to freely copy, distribute, and use
   this file, under the following conditions:
      - No changes are made.
      - No direct commercial advantage is obtained.
      - No liability is attributed to the author for any damages incurred.
  -------------------------------------------------------------------------*/

/*************************************************************************
 * This file defines a C function fft that, by calling another function       *
 * fft_rec (also defined), calculates an FFT recursively.  Usage:             *
 *   fft(N, x, X);                                                            *
 * Parameters:                                                                *
 *   N: number of points in FFT (must equal 2^n for some integer n >= 1)      *
 *   x: pointer to N time-domain samples given in rectangular form (Re x,     *
 *      Im x)                                                                 *
 *   X: pointer to N frequency-domain samples calculated in rectangular form  *
 *      (Re X, Im X)                                                          *
 * Similarly, a function ifft with the same parameters is defined that        *
 * calculates an inverse FFT (IFFT) recursively.  Usage:                      *
 *   ifft(N, x, X);                                                           *
 * Here, N and X are given, and x is calculated.                              *
 *************************************************************************/

#include <stdlib.h>
#include <math.h>

/* macros */
#define TWO_PI (6.2831853071795864769252867665590057683943L)
```

```
/* function prototypes */
void fft(int N, double (*x)[2], double (*X)[2]);
void fft_rec(int N, int offset, int delta, double (*x)[2], double (*X)[2], double (*XX)[2]);
void ifft(int N, double (*x)[2], double (*X)[2]);

/* FFT */
void fft(int N, double (*x)[2], double (*X)[2])
{
  /* Declare a pointer to scratch space. */
  double (*XX)[2] = malloc(2 * N * sizeof(double));

  /* Calculate FFT by a recursion. */
  fft_rec(N, 0, 1, x, X, XX);

  /* Free memory. */
  free(XX);
}
/* FFT recursion */
void fft_rec(int N, int offset, int delta,
             double (*x)[2], double (*X)[2], double (*XX)[2])
{
  int N2 = N/2;            /* half the number of points in FFT */
  int k;                   /* generic index */
  double cs, sn;           /* cosine and sine */
  int k00, k01, k10, k11;  /* indices for butterflies */
  double tmp0, tmp1;       /* temporary storage */

  if(N != 2)  /* Perform recursive step. */
    {
      /* Calculate two (N/2)-point DFT's. */
      fft_rec(N2, offset, 2*delta, x, XX, X);
      fft_rec(N2, offset+delta, 2*delta, x, XX, X);

      /* Combine the two (N/2)-point DFT's into one N-point DFT. */
      for(k=0; k<N2; k++)
        {
          k00 = offset + k*delta;    k01 = k00 + N2*delta;
```

```
          k10 = offset + 2*k*delta;  k11 = k10 + delta;
          cs = cos(TWO_PI*k/(double)N); sn = sin(TWO_PI*k/(double)N);
          tmp0 = cs * XX[k11][0] + sn * XX[k11][1];
          tmp1 = cs * XX[k11][1] - sn * XX[k11][0];
          X[k01][0] = XX[k10][0] - tmp0;
          X[k01][1] = XX[k10][1] - tmp1;
          X[k00][0] = XX[k10][0] + tmp0;
          X[k00][1] = XX[k10][1] + tmp1;
        }
    }
  else  /* Perform 2-point DFT. */
    {
      k00 = offset; k01 = k00 + delta;
      X[k01][0] = x[k00][0] - x[k01][0];
      X[k01][1] = x[k00][1] - x[k01][1];
      X[k00][0] = x[k00][0] + x[k01][0];
      X[k00][1] = x[k00][1] + x[k01][1];
    }
}
/* IFFT */
void ifft(int N, double (*x)[2], double (*X)[2])
{
  int N2 = N/2;       /* half the number of points in IFFT */
  int i;              /* generic index */
  double tmp0, tmp1;  /* temporary storage */

  /* Calculate IFFT via reciprocity property of DFT. */
  fft(N, X, x);
  x[0][0] = x[0][0]/N;    x[0][1] = x[0][1]/N;
  x[N2][0] = x[N2][0]/N;  x[N2][1] = x[N2][1]/N;
  for(i=1; i<N2; i++)
    {
      tmp0 = x[i][0]/N;       tmp1 = x[i][1]/N;
      x[i][0] = x[N-i][0]/N;  x[i][1] = x[N-i][1]/N;
      x[N-i][0] = tmp0;       x[N-i][1] = tmp1;
    }
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. PLOTTING M CODE

```
%% %%%%%%%%%%%%%%%%%%%%% mplot.m %%%%%%%%%%%%%%%%%%%%%%% %%
% This m-file takes the resulting .mat file from LPIT and  %
% runs the Choi-Williams Distribution (CWD) on N samples   %
% of the file. N must be a power of 2 and is typically 256 %
% or 512. These results are then plotted.                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%clear, clc, close all

%% %%%%%%%%%%%%%%%%%%%% Load File %%%%%%%%%%%%%%%%%%%%% %%
% Load data file for analysis
%%FMCW F_1_7_250_20_0
%load H:\Thesis\Choi\Test_signals\F_1_7_250_20_0.mat
%fs = 7000;
%axis([0 0.07 500 1500]);

%%%%%% For Testing %%%%%%%%%%%%%%%%%%%%%%%%
%%PSK (Frank) FR_1_7_8_1_0
%load H:\Thesis\Choi\FR_1_7_8_1_0.mat
%fs = 7000;
%axis([0 0.07 500 1500]);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%PSK (Frank) FR_1_7_4_1_0
%load H:\Thesis\Choi\Test_signals\FR_1_7_4_1_0.mat
%fs = 7000;
%axis([0 0.07 500 1500]);

%%FSK (Costas)C_1_15_5000_0
load H:\Thesis\Choi\Test_signals\C_1_15_5000_0.mat
fs = 15057;
%axis([0 0.033 0 7500]);

%%FSK/PSK FSK_PSK_Costas_15_5_0
%load H:\Thesis\Choi\Test_signals\FSK_PSK_Costas_15_5_0.mat
%fs = 15057;
%axis([0 0.033 0 7500]);

%% %%%%%%%%%%%%%%% Set Up Variables %%%%%%%%%%%%%%%%%%% %%
N = 512;
I = I(1:N);
Q = Q(1:N);
x=(I+j*Q)';
c=length(x);

%% %%%%%%%%%%%%%% Set Up for Plot Functions %%%%%%%%%%%%%% %%
T=1/fs;
freqp=0:fs/c:fs-fs/c;          %% Frequency
freqp=freqp/2;
time=0:T:c*T-T;                %% Time
```

```
%% %% Run the Choi Williams Distribution Computation %% %%
t1=clock;   %% This starts the timer
W = choiwilliams(x);
t2=clock;   %% This stops the timer
disp('Total Time')
e = etime(t2, t1)
%% %%%%%%%%%%%%%%%%%% Plot Results %%%%%%%%%%%%%%%%%% %%
%figure(14)
%mesh(time,freqp,W); shading interp;
%xlabel('Time (s)');
%ylabel('Frequency (Hz)');
%zlabel('Magnitude (V)');
%view(90,0)

%figure(15)
%mesh(time,freqp,W); shading interp;
%xlabel('Time (s)');
%ylabel('Frequency (Hz)');
%zlabel('Magnitude (V)');
%view(0,0)

%figure(16)
%mesh(time,freqp,W); shading interp;
%xlabel('Time (s)');
%ylabel('Frequency (Hz)');
%zlabel('Magnitude (V)');

figure(17)
mesh(time,freqp,W); shading interp; box on;
xlabel('Time (s)');
ylabel('Frequency (Hz)');
zlabel('Magnitude (V)');
view(0,90)
% The axis may have to be adjusted based on the LPI signal being
analyized.
axis([0 0.033 0 7500]);

%figure(18)
% This figure is generated using the contour command instead of the
mesh
% command which generates a slightly different graph.
%maxi = max(max(W));
%mini = min(min(W));
%lev = linspace(mini,maxi,65);
%contour(time,freqp,W,lev)
%ylabel('Frequency (Hz)');
%xlabel('Time (s)');
%zlabel('Magnitude (V)');
%axis([0 0.033 0 7500]);

% End M-File
```

```
%% %%%%%%%%%%%%%%%%%%%% cplot.m %%%%%%%%%%%%%%%%%%%%%% %%
% This m-file takes the resulting .txt file from the       %
% the Choi-Williams Distitribution (CWD) of N samples.     %
% The results are then plotted using the mesh and contour  %
% functions.                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear, clc, close all

N=512;
%% %%%%%%%%%%%%%%%%%%%% Load File %%%%%%%%%%%%%%%%%%%%%% %%
% The txt file must be changed for each C code output.

%% FMCW F_1_7_250_20_0
%load -ascii H:\Thesis\Choi\C_output\F_1_7_250_20_0_Cout_512SRC.txt
%fs = 7000;
%W=N*F_1_7_250_20_0_Cout_512SRC;

%% PSK (Frank) FR_1_7_4_1_0
%load -ascii H:\Thesis\Choi\C_output\FR_1_7_4_1_0_Cout_512SRC.txt
%fs = 7000;
%W=N*FR_1_7_4_1_0_Cout_512SRC;

%% FSK (Costas)C_1_15_5000_0
%load -ascii H:\Thesis\Choi\C_output\C_1_15_5000_0_Cout_512SRC.txt
%fs = 15057;
%W=N*C_1_15_5000_0_Cout_512SRC;
%axis([0 0.033 0 7500]);

%% FSK/PSK FSK_PSK_Costas_15_5_0
load -ascii
H:\Thesis\Choi\C_output\FSK_PSK_Costas_15_5_0_Cout_512SRCHW.txt
fs = 15057;
W=N*FSK_PSK_Costas_15_5_0_Cout_512SRCHW;
%axis([0 0.033 0 7500]);

%% %%%%%%%%%%%%% Set Up for Plot Functions %%%%%%%%%%%%% %%
c=length(W);

T=1/fs;
freqp=0:fs/c:fs-fs/c;          %% Frequency
freqp=freqp/2;
time=0:T:c*T-T;                %% Time

%% %%%%%%%%%%%%%%%%%%%% Plot Results %%%%%%%%%%%%%%%%%%%% %%
%figure(14)
%mesh(time,freqp,W); shading interp;
%xlabel('Time (s)');
%ylabel('Frequency (Hz)');
%zlabel('Magnitude (V)');
%view(90,0)
```

```
%figure(15)
%mesh(time,freqp,W); shading interp;
%xlabel('Time (s)');
%ylabel('Frequency (Hz)');
%zlabel('Magnitude (V)');
%view(0,0)

%figure(16)
%mesh(time,freqp,W); shading interp;
%xlabel('Time (s)');
%ylabel('Frequency (Hz)');
%zlabel('Magnitude (V)');

figure(17)
mesh(time,freqp,W); shading interp; box on;
xlabel('Time (s)');
ylabel('Frequency (Hz)');
zlabel('Magnitude (V)');
view(0,90)
% The axis may have to be adjusted based on the LPI signal being
analyized.
axis([0 0.033 0 7500]);

%figure(8)
%maxi = max(max(W));
%mini = min(min(W));
%lev = linspace(mini,maxi,65);
%contour(time,freqp,W,lev)
%ylabel('Frequency (Hz)');
%xlabel('Time (s)');
%zlabel('Magnitude (V)');
%title('Contour of CWD')

% End M-File
```

APPENDIX E. SRC MAIN CODE

All user defined variables were included in a separate file. Only the user defined file must be altered to change the vector length. Note that N and n2 are paired together.

```
#define PI 3.14159265358979323846264338327795

#define N  512
#define n2 9

//#define N  256
//#define n2 8
```

```
#include <stdio.h>
#include <math.h>
#include <map.h>
#include <time.h>
#include <libmap.h>
#include "userdef.c"
#include "high_prec_time.c"

/******* Choi Williams Distribution Computation ********
*******************************************************
* N = length of vector (file) Must be an Integer power *
*     of two (256 or 512 is recommended).              *
*                                                      *
* *data = pointer to input data                        *
* Data must be I and Q data in two respective columns  *
*                                                      *
* Output vector will be NxN                            *
* Rows represent change in time                        *
* Columns represent change in frequency                *
*******************************************************/

void fft_map (float input[],float twiddle[],
              float output[],int n,int frflag,
              int64_t* t_dma, int64_t* t_fwd,
              int64_t* t_call, int map);

int main()
{
 /* Define Variables */
 int i,j,k,r,q,P,L,M,n,a,b,nmap;

 double sigma,W,cn;
 float  Iin[N], Qin[N];

 /* Define Pointers */
 FILE *data;

 double (*IData),          (*QData);
```

```
 double (*Ishift),         (*Qshift);
 double (*IX1),            (*QX1);
 double (*IX2),            (*QX2);
 double (*Isum),           (*Qsum);
 double (*IWV)[N],         (*QWV)[N];
 double (*IWVweight)[N-1], (*QWVweight)[N-1];
 double (*Ikern)[N],       (*Qkern)[N];
 double (*IDataOut)[N],    (*QDataOut)[N];
 double (*mu),             (*weight)[N];

/****************** FFT Variables ******************
 These variables are only used for the FFT.          */

int   frflag, nbytes, npoint;
float       *twiddle;
float       (*tempin), (*tempout);
double      rad;

/***************** Timing Variables ******************
 These variables are only used for timing.           */

 struct timeval tottime, stime, etime;
 struct timeval totfftime, sfftime, efftime;

 float tt,fftime,time1,time2;

 int64_t t_dma, t_fwd, t_call;
 float      ft_dma, ft_fwd, ft_call, ftot_dma, ftot_fwd, ftot_call;

 fftime   = 0;
 ftot_dma  = 0;
 ftot_fwd  = 0;
 ftot_call = 0;

/*************** Initialize Variables ****************
*****************************************************/
 k = 1;
 j = 0;
 P = N/2;
 M = (2*P)-1;
 a = -P;
 b = P-1;

/****************** Sigma ***************************
*****************************************************
Can use to change the weight of the Choi Williams
distribution. A large sigma will result in a
Wigner-Ville Distribution.                           */

sigma = 1;

/***************** Allocate memory ******************/
     IData = Cache_Aligned_Allocate(N*sizeof(double));
     QData = Cache_Aligned_Allocate(N*sizeof(double));
```

```
/****************** Open Data ************************
*****************************************************
Default: Data is stored in a file called
              Test_signals_txt                        */

 data  = fopen("Test_signals_txt/INPUT.txt","r");

 if(data == NULL)
  {
   puts("Error opening file.");
   return(1);
  }
  fscanf(data,"%f", &Iin[0]);
  IData[0] = Iin[0];

  for(i=0; i<((2*N)-1);i++)
   {
    if(i % 2 == 1) /*Odd data is QData*/
     {
      fscanf(data, "%f", &Iin[k]);
      IData[k] = Iin[k];
      k++;
     }
    else
     {
      fscanf(data,"%f", &Qin[j]);
      QData[j] = Qin[j];
      j++;
     }
   }

/* Close the file */
fclose(data);

/************** Start Time the SRC Code ***************
 The total time will be from this point to the stage  *
 right before the output to txt. This is to make as   *
 fair a comparison between the SRC code, the C code   *
 and the MATLAB code.                                 */

 gettimeofday(&stime,NULL);

/***** Compute the Wigner-Ville Distribution ***********
*****************************************************
The Wigner-Ville Distribution can be used to
computer the Choi-Williams Distribution. The
variables IWV and QWV are the NxN matrix output.

First, the data is shifted.
Second, the data is broken into X1 and X2.
Lastly, the data is recombined into IWV and QWV.       */

    /*************** Allocate memory ******************/
     Ishift = Cache_Aligned_Allocate(N*sizeof(double));
     Qshift = Cache_Aligned_Allocate(N*sizeof(double));
```

```
     IX1    = Cache_Aligned_Allocate(N*sizeof(double));
     QX1    = Cache_Aligned_Allocate(N*sizeof(double));
     IX2    = Cache_Aligned_Allocate(N*sizeof(double));
     QX2    = Cache_Aligned_Allocate(N*sizeof(double));
     IWV    = Cache_Aligned_Allocate(N*N*sizeof(double));
     QWV    = Cache_Aligned_Allocate(N*N*sizeof(double));
     Ikern  = Cache_Aligned_Allocate(N*N*sizeof(double));
     Qkern  = Cache_Aligned_Allocate(N*N*sizeof(double));

for(i=a; i<b+1; i++)
 {
/********************* Shift ***************************
 Shift both I and Q                                    */

  if(i>= 0)
   {
    for(k=0; k<i; k++)
     {
      Ishift[k] = 0;
      Qshift[k] = 0;
     }
    for (k=i; k<N; k++)
     {
      Ishift[k] = IData[k-i];
      Qshift[k] = QData[k-i];
     }
   }
 else
   {
    for(k=0; k<N+i; k++)
     {
      Ishift[k] = IData[k-i];
      Qshift[k] = QData[k-i];
     }
    for(k=N+i; k<N; k++)
     {
      Ishift[k] = 0;
      Qshift[k] = 0;
     }
   }
/***************** End Shift ***************************/
/************* Determine X1 and X2 **********************/
 for(k=0; k<P; k++)
  {
   IX1[k] = Ishift[P+k-1];
   QX1[k] = Qshift[P+k-1];
   IX2[k] = Ishift[P-1-k];
   QX2[k] = -Qshift[P-1-k];
  }

 IX1[k] = 0;
 QX1[k] = 0;
 IX2[k] = 0;
 QX2[k] = 0;
```

```
 for(k=(P+1); k<N; k++)
  {
   IX1[k] = Ishift[k-P-1];
   QX1[k] = Qshift[k-P-1];
   IX2[k] = Ishift[M+(M-P+1)-k];
   QX2[k] = -Qshift[M+(M-P+1)-k];
  }

/****************** End X1 and X2 *********************/
/******************** Determine WV ********************/
 for (k=0; k<N; k++)
  {
   IWV[i+P][k] = ((IX1[k]*IX2[k])-(QX1[k]*QX2[k]));
   QWV[i+P][k] = -((IX1[k]*QX2[k])+(IX2[k]*QX1[k]));
  }

/********************* End WV ************************/
 }

    /*************** Release memory ******************/
     free(IX2);
     free(QX2);
     free(IX1);
     free(QX1);
     free(Ishift);
     free(Qshift);

/********** End Wigner-Ville Distribution *************/
/****** Compute the Choi-Williams Distribution *********
*****************************************************
The Choi-Williams kernel is a summation of weighted WV
outputs.                                              */

/****************** Determine mu *********************
 Mu is used for array indexing in determining the
 weighting function.                                  */

    /*************** Allocate memory ******************/
     mu = Cache_Aligned_Allocate(N*sizeof(double));

for(k=0; k<N; k++)
 mu[k]=b-k;

/******************** End mu ************************/
/************* Determine weight ********************
 These nested for loops determine the weighting
 function in the Choi-Williams Distribution.          */

    /*************** Allocate memory ******************/
     weight    = Cache_Aligned_Allocate(N*N*sizeof(double));
     Isum      = Cache_Aligned_Allocate(N*sizeof(double));
     Qsum      = Cache_Aligned_Allocate(N*sizeof(double));
     IWVweight = Cache_Aligned_Allocate(N*N*sizeof(double));
     QWVweight = Cache_Aligned_Allocate(N*N*sizeof(double));
```

```
 cn = sigma/(4*PI);

  for(L=0; L<N; L++)
   {
    for(n=1; n<P; n++)
     {
      for(i=0; i<N; i=i+4)
       {
        W = sqrt(cn/(n*n))
            *exp((-((mu[i]-(L-P))*(mu[i]-(L-P)))*sigma/(4*(n*n))));
        weight[i][n] = W;
        weight[i][N-n] = W;

        W = sqrt(cn/(n*n))
            *exp((-((mu[i+1]-(L-P))*(mu[i+1]-(L-P)))*sigma/(4*(n*n))));
        weight[i+1][n] = W;
        weight[i+1][N-n] = W;

        W = sqrt(cn/(n*n))
            *exp((-((mu[i+2]-(L-P))*(mu[i+2]-(L-P)))*sigma/(4*(n*n))));
        weight[i+2][n] = W;
        weight[i+2][N-n] = W;

        W = sqrt(cn/(n*n))
            *exp((-((mu[i+3]-(L-P))*(mu[i+3]-(L-P)))*sigma/(4*(n*n))));
        weight[i+3][n] = W;
        weight[i+3][N-n] = W;
       } //End i loop
     }//End n loop

    for(j=0; j<N-1; j++) //columns
     {
      Isum[j] = 0;
      Qsum[j] = 0;
      for(k=0; k<N; k++)//rows
       {
        IWVweight[k][j] = weight[k][j+1] * IWV[k][j+1];
        QWVweight[k][j] = weight[k][j+1] * QWV[k][j+1];
        Isum[j] = Isum[j] + IWVweight[k][j];
        Qsum[j] = Qsum[j] + QWVweight[k][j];
       } //End k loop
     } //End j loop

/****************** End weight ***********************/
/*************** Determine kernel ********************/

   for(r=0; r<N; r++)
    {
     //printf("L = %i; r = %i; N=%i\n",L,r,N);
     if(r==0)
       {
        Ikern[r][L] = IWV[N-L-1][r];
        Qkern[r][L] = 0;
       }
     else
```

```
       {
        Ikern[r][L] = Isum[r-1];
        Qkern[r][L] = -Qsum[r-1];
       }
     }

 } //End L loop

    /*************** Release memory ******************/
     free(IWVweight);
     free(QWVweight);
     free(Isum);
     free(Qsum);
     free(weight);
     free(mu);
     free(IWV);
     free(QWV);

/********** End Choi-Williams Distribution *************/
/***************** Determine FFT **********************
*******************************************************
The FFT algorithm is for an Nx1 array so it used in a
loop to calculate the FFT for an NxN array.           */

    /*************** Allocate memory ******************/
     frflag = 1;  //does an FFT (vs. an IFFT)
     npoint = 1 << n2;
     nbytes = npoint*8;

     tempin   = (float *)Cache_Aligned_Allocate(nbytes);
     tempout  = (float *)Cache_Aligned_Allocate(nbytes);
     IDataOut = Cache_Aligned_Allocate(N*N*sizeof(double));
     QDataOut = Cache_Aligned_Allocate(N*N*sizeof(double));

/*************** Build Twiddle Table ******************
This table is used in the FFT.                        */

   nbytes = npoint*4;
   twiddle = (float *)Cache_Aligned_Allocate(nbytes);

   i=0;
   for(j=0; j< npoint/2; j++)
     {
      rad = 2.0*PI*((double)j/(double)npoint);
      twiddle[i] = cos(rad);
      twiddle[i+1] = -sin(rad);
      i+=2;
     }

/**************** End Twiddle Table *******************/
/****************** Allocate MAP **********************
This is to allocate the MAP.                          */

nmap = 1;
if (map_allocate (nmap))
```

```
 {
  fprintf (stdout, "Map allocation falied.\n");
  exit(1);
 }
nmap = 0;

/*************** End Allocate MAP ********************/
for (j=0; j<N; j++)
 {

/*************** Arrange Data for FFT *****************
 The data needs to be in [i,q,i,q,etc.] format.        */
  k= 0;
  for(i=0;i<N;i++)
   {
    tempin[k] = Ikern[i][j];
    tempin[k+1] = Qkern[i][j];
    k+=2;
   }

/*************** Perform FFT Algorithm ***************/

   gettimeofday(&sfftime,NULL);

   fft_map(tempin, twiddle, tempout, n2, frflag,
            &t_dma, &t_fwd, &t_call, nmap);

   gettimeofday(&effttime,NULL);

   tt = timeval_subtract(&totfftime, &effttime, &sfftime);
   time2 = totfftime.tv_sec + totfftime.tv_usec*1e-6;
   time1 = time2+fftime;
   fftime = time1;

    ft_dma  = t_dma  *1e-8;
    ft_fwd  = t_fwd  *1e-8;
    ft_call = t_call *1e-8;
    ftot_dma = ftot_dma + ft_dma;
    ftot_fwd = ftot_fwd + ft_fwd;
    ftot_call = ftot_call + ft_call;

/************* Arrange Data for Output ****************
 The data needs to be in N X N format.                 */
  k=0;
  for (i=0; i<N; i++)
   {
    IDataOut[i][j] = 2*tempout[k];
    QDataOut[i][j] = 2*tempout[k+1];
    k+=2;
   }
 } // End j loop

/******************** Free MAP ***********************
 This is to free the MAP.                              */
nmap = 1;
```

```
if (map_free (nmap))
 {
  fprintf (stdout, "Map deallocation falied.\n");
  exit(1);
 }

/****************** End Free MAP ***********************/
/***************** Release memory **********************/
     free(tempin);
     free(tempout);
     free(Ikern);
     free(Qkern);

/********************** End FFT ************************/
/*************** End Time the SRC Code *****************
 The total time will be from this point to the stage   *
 right before the output to txt. This is to make as    *
 fair a comparison between the SRC code, the C code    *
 and the MATLAB code.                                  */

 gettimeofday(&etime, NULL);
 tt = timeval_subtract(&tottime, &etime, &stime);
 time1 = tottime.tv_sec+ tottime.tv_usec*1e-6;

 printf("Total SRC time = %10.8f seconds.\n", time1);
 printf("Total MAP time = %10.8f seconds \n", ftot_call);
 printf("      DMA time = %10.8f seconds \n", ftot_dma);
 printf("      FFT time = %10.8f seconds \n", ftot_fwd);
 printf("     Call Time = %10.8f seconds.\n", ffttime - ftot_call);

 printf("Total time excluding data loads was %4.6f seconds.\n", time1 -
(ffttime-ftot_call));

/*************** Output Data ************************
*****************************************************
This outputs the final result to a text file called
out.txt that can be pulled into MATLAB and graphed.    */
 data  = fopen("out.txt","w");

if(data == NULL)
 {
  puts("Error creating file.");
  return(1);
 }
for (r=0; r<N; r++)
 {
  for (q=0; q<N-1; q++)
    fprintf(data, "%12f\t", IDataOut[r][q]);
    fprintf(data, "%12f\n", IDataOut[r][q]);
 }

/* Close the file */
fclose(data);

puts("Data saved to out.txt");
```

```
/***************** End Output *************************
*****************************************************/

/***************** Release memory **********************/
free(IData);
free(QData);
free(IDataOut);
free(QDataOut);

return(0);
}
```

APPENDIX F. MAP ROUTINE .MC CODE

```
#include <libmap.h>
void cfft_fp32(int n, int inv, int64_t a_in, int64_t b_in,
               int64_t c_in, int64_t d_in, int sig_valid,
               int64_t e_in, int64_t f_in, int twid_valid,
               int starting, int64_t *a_out, int64_t *b_out,
               int64_t *c_out, int64_t *d_out,
               int *data_valid_out, int *xfrm_addr_out);

void fft_map(float input[], float twiddle[], float output[],
             int n, int frflag,int64_t* t_dma, int64_t* t_fwd,
             int64_t* t_call, int map)
{
      int         i, starting, inv, loop, npoint;
      int         nbytes, cm_loc, obm_loc;
      int64_t     t0, t1, t2, t3, t4;

int   inidx, outidx, outctr, twididx, sig_len, valid_in, valid_out;
      int         xfrm_addr_out;
int64_t     a_in,  b_in,  c_in,  d_in,  e_in,  f_in,  a_out,  b_out,  c_out,
d_out;

      /* input and output */
      OBM_BANK_A (a, int64_t, MAX_OBM_SIZE)
      OBM_BANK_B (b, int64_t, MAX_OBM_SIZE)
      OBM_BANK_C (c, int64_t, MAX_OBM_SIZE)
      OBM_BANK_D (d, int64_t, MAX_OBM_SIZE)
      /* twiddle table */
      OBM_BANK_E (e, int64_t, MAX_OBM_SIZE)
      OBM_BANK_F (f, int64_t, MAX_OBM_SIZE)

      /* start MAP timing */
      start_timer();
      read_timer(&t0);

      /*  move twiddle table */
      npoint = 1 << n;
      nbytes = npoint*4;
      DMA_CPU(CM2OBM,e, MAP_OBM_stripe(1,"E,F"), twiddle, 1, nbytes, 0);
      wait_DMA(0);

      /*  move input data   */
      nbytes = npoint*8;
      obm_loc = 0;
      cm_loc  = 0;
DMA_CPU(CM2OBM, &a[obm_loc], MAP_OBM_stripe(1,"A,B,C,D"),
  &input[cm_loc], 1, nbytes, 0);
      wait_DMA(0);
      read_timer(&t1);

      /* do fft */
      inv = 0;
```

```
      sig_len = npoint/4;
      for (loop=0; loop < frflag; ++loop) {
            inidx = 0;
            outidx = 0;
            outctr = 0;
            starting = 1;
#pragma loop noloop_dep
#pragma loop noldst_clsh
            do {
                  valid_in = ( inidx < sig_len) ? 1 : 0;

                  a_in = a[inidx];
                  b_in = b[inidx];
                  c_in = c[inidx];
                  d_in = d[inidx];
                  e_in = e[inidx];
                  f_in = f[inidx];

                  ++inidx;

cfft_fp32 (n, inv, a_in, b_in, c_in, d_in, valid_in,
           e_in, f_in, valid_in, starting, &a_out,
           &b_out, &c_out, &d_out, &valid_out, &outidx);

                  if ( valid_out ) {
                        a[outidx] = a_out;
                        b[outidx] = b_out;
                        c[outidx] = c_out;
                        d[outidx] = d_out;
                  }
                  cg_accum_add_32_np(1, valid_out, 0, starting,
 &outctr);
                  starting = 0;
            } while ( outctr < sig_len );
            if ( loop == 0 ) {
                  inv = 1;
                  read_timer(&t2);
            }
      }
      read_timer(&t3);

      /*  move output data   */
      nbytes = npoint*8;
      obm_loc = 0;
      cm_loc  = 0;
      DMA_CPU(OBM2CM, &a[obm_loc], MAP_OBM_stripe(1,"A,B,C,D"),
&output[cm_loc], 1, nbytes, 0);
      wait_DMA(0);
      read_timer(&t4);

      *t_dma = (t1 - t0) + (t4 - t3);
      *t_fwd = t2 - t1;
      *t_call = t4 - t0;
}
```

APPENDIX G. MAKEFILE CODE

```
# ----------------------------------
# User defines FILES, MAPFILES, and BIN here
# ----------------------------------
FILES           = choiSRC.c

MAPFILES        = fft.mc

BIN             = Final_Output

SRC_VERSION = comp
SRC_TARGET = map_e
SRC_FFT_LIB = /opt/SRCCI2.2/fft_lib

# ----------------------------------
# Multi chip info provided here
# (Leave commented out if not used)
# ----------------------------------
#PRIMARY        = <primary file 1>   <primary file 2>

#SECONDARY      = <secondary file 1> <secondary file 2>

#CHIP2          = <file to compile to user chip 2>

#----------------------------------
# User defined directory of code routines
# that are to be inlined
#----------------------------------

#INLINEDIR      =

# ----------------------------------
# User defined macros info supplied here
#
# (Leave commented out if not used)
# ----------------------------------
#MACROS         = <directory-name/macro-file>

#MY_BLKBOX      = <directory-name/blackbox-file>
#MY_NGO_DIR     = <directory-name>
#MY_INFO        = <directory-name/info-file>
# ----------------------------------
# Floating point macros selection
# ----------------------------------

#FPMODE         = SRC_IEEE_V1 # Default SRC version IEEE
#FPMODE         = SRC_IEEE_V2 # Size reduced SRC IEEE with
                               # special rounding mode
# ----------------------------------
# User supplied MCC and MFTN flags
# ----------------------------------
```

```
MY_MCCFLAGS     = -log
MY_MFTNFLAGS    = -log

# ----------------------------------
# User supplied flags for C & Fortran compilers
# ----------------------------------

CC              = icc   # icc   for Intel cc for Gnu
FC              = ifort # ifort for Intel f77 for Gnu
#LD             = ifort # for Fortran or C/Fortran mixed
LD              = icc   # for C codes

CFLAGS          =       -O3 –tpp7 –xW -ip
MY_FFLAGS       =
MY_LDFLAGS      =       # Flags to include libs if needed
# ----------------------------------
# VCS simulation settings
# (Set as needed, otherwise just leave commented out)
# ----------------------------------

USER_MACROLIBS = $(SRC_FFT_LIB)
PAR_OPTIONS = -t 50

#USEVCS         = yes   # YES or yes to use vcs instead of vcsi
#VCSDUMP        = yes   # YES or yes to generate vcd+ trace dump
# ----------------------------------
# No modifications are required below
# ----------------------------------
MAKIN   ?= $(MC_ROOT)/opt/srcci/comp/lib/AppRules.make
include $(MAKIN)
```

APPENDIX H. TIMING CODE

This code was used to generate timing with more time resolution that whole seconds. The file name had to be included in the main SRC program.

```
int timeval_subtract (struct timeval *result, struct timeval *x, struct
timeval *y);

/* Subtract the `struct timeval' values X and Y,
        storing the result in RESULT.
        Return 1 if the difference is negative, otherwise 0.  */

     int
     timeval_subtract (result, x, y)
          struct timeval *result, *x, *y;
     {
       /* Perform the carry for the later subtraction by updating y. */
       if (x->tv_usec < y->tv_usec) {
         int nsec = (y->tv_usec - x->tv_usec) / 1000000 + 1;
         y->tv_usec -= 1000000 * nsec;
         y->tv_sec += nsec;
       }
       if (x->tv_usec - y->tv_usec > 1000000) {
         int nsec = (x->tv_usec - y->tv_usec) / 1000000;
         y->tv_usec += 1000000 * nsec;
         y->tv_sec -= nsec;
       }

       /* Compute the time remaining to wait.
          tv_usec is certainly positive. */
       result->tv_sec = x->tv_sec - y->tv_sec;
       result->tv_usec = x->tv_usec - y->tv_usec;

       /* Return 1 if result is negative. */
       return x->tv_sec < y->tv_sec;
     }
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX I. TIME TRIALS

	FMCW								
	MATLAB Code[1]			MATLAB Code[2]			C Code		
Trials	Total (sec)	CWD Kernel (sec)	FFT (sec)	Total (sec)	CWD Kernel (sec)	FFT (sec)	Total (sec)	FFT (sec)	CWD Kernel (sec)
1	42.135000	33.375445	0.032994	33.875000	27.220885	0.021414	6.808524	0.080723	6.619582
2	42.931000	34.054506	0.033048	33.829000	27.249300	0.021069	6.825237	0.081218	6.635985
3	42.010000	33.249822	0.034361	33.937000	27.271555	0.021729	6.844558	0.084141	6.651393
4	41.900000	33.164924	0.033571	34.078000	27.493605	0.021720	6.800048	0.080538	6.611869
5	41.978000	33.271557	0.032987	33.890000	27.292776	0.020370	6.879942	0.080919	6.690270
6	42.666000	33.748402	0.033775	33.922000	27.243836	0.019518	6.856262	0.080819	6.667106
7	43.104000	34.183792	0.032832	34.579000	27.878681	0.026133	6.807355	0.080926	6.617987
8	43.401000	34.476949	0.033467	34.234000	27.504892	0.021777	6.882586	0.080646	6.693875
9	43.291000	34.353970	0.034650	34.156000	27.482425	0.020707	6.850605	0.080853	6.661507
10	42.150000	33.368591	0.032983	34.188000	27.491442	0.021798	6.823736	0.080805	6.634692
11	42.088000	33.319626	0.033798	33.844000	27.205099	0.021059	6.881128	0.081181	6.691346
12	42.150000	33.375786	0.032994	33.953000	27.345540	0.021920	6.863357	0.080828	6.674034
13	42.103000	33.342221	0.033278	33.860000	27.228194	0.021159	6.887703	0.081034	6.698063
14	42.103000	33.346012	0.032654	33.937000	27.316089	0.021754	6.870801	0.081149	6.681285
15	42.197000	33.412101	0.033251	34.375000	27.663710	0.021993	6.860159	0.082067	6.669467
16	42.182000	33.382078	0.033812	34.500000	27.737586	0.022044	6.847492	0.081104	6.657776
17	42.166000	33.377174	0.033267	33.937000	27.250251	0.022059	6.853079	0.080427	6.664381
18	42.166000	33.400553	0.034492	34.047000	27.350141	0.021588	6.834894	0.081031	6.645633
19	42.166000	33.400032	0.033077	33.984000	27.329706	0.021966	6.883038	0.081108	6.693102
20	42.275000	33.510122	0.034383	34.438000	27.492956	0.021777	6.885325	0.079861	6.697062
Total	42.36	33.56	0.03	34.08	27.40	0.02	6.85	0.08	6.66

All timing was done in seconds.

C compiler options = -O3 -tpp7 -xW - align -Zp16 -ipo -static

	FMCW						
	SRC Code						
Trials	Total	Total*	MAP Time	DMA time	FFT time	Call Time	uP Time
1	39.539112	39.335979	0.03088881	0.02640878	0.00441859	0.20313405	39.30508923
2	36.850487	36.647186	0.02943201	0.02495195	0.00441859	0.20329989	36.61775486
3	37.895901	37.682663	0.03158952	0.02710949	0.00441859	0.21323718	37.65107403
4	37.156990	36.913876	0.02748343	0.02300341	0.00441859	0.24311545	36.88639117
5	36.410980	36.200962	0.02978616	0.02530614	0.00441859	0.21001655	36.17117751
6	36.181431	35.977699	0.02488480	0.02040478	0.00441859	0.20373017	35.95281585
7	36.245834	36.042522	0.02501928	0.02053926	0.00441859	0.20331076	36.01750431
8	36.527813	36.324909	0.03044737	0.02596732	0.00441859	0.20290345	36.29446214
9	36.311993	36.106174	0.02939976	0.02491973	0.00441859	0.20581640	36.07677649
10	36.065289	35.860901	0.02539677	0.02091678	0.00441859	0.20438816	35.83550361
11	36.206989	36.004547	0.02463551	0.02015550	0.00441859	0.20244157	35.97991221
12	36.407448	36.199394	0.02469767	0.02021767	0.00441859	0.20805340	36.17469674
13	36.246040	36.031265	0.03049769	0.02601764	0.00441859	0.21477588	36.00076677
14	36.433971	36.222206	0.02617408	0.02169408	0.00441859	0.21176679	36.19603054
15	36.422344	36.214184	0.02680014	0.02232016	0.00441859	0.20816153	36.18738254
16	36.508163	36.298981	0.02438374	0.01990375	0.00441859	0.20918222	36.27459749
17	36.360844	36.156540	0.02582351	0.02134350	0.00441859	0.20430477	36.13071538
18	36.495571	36.292130	0.02489622	0.02041622	0.00441859	0.20344077	36.26723415
19	36.250816	36.046371	0.03016247	0.02568245	0.00441859	0.20444332	36.01621056
20	36.344505	36.128628	0.03166118	0.02718118	0.00441859	0.21587856	36.09696557
Total	36.64	36.43	0.028	0.023	0.004	0.209	36.407

All timing was done in seconds.

* Note: The time to load data to the MAP was removed.

MAP Time: Time spent on the MAP

DMA Time: Time spend moving data to and from the MAP

FFT Time: Time spent performing the FFT

SRC*: SRC Time - DMA Time - Call Time

uP Time: SRC Code - MAP Time - Call Time

SRC CFLAGS = -O3 -tpp7 -xW –ip

	Frank Code								
	MATLAB Code1			MATLAB Code2			C Code		
Trials	Total (sec)	CWD Kernel (sec)	FFT (sec)	Total (sec)	CWD Kernel (sec)	FFT (sec)	Total (sec)	FFT (sec)	CWD Kernel (sec)
1	42.369000	33.365577	0.033414	34.000000	27.315711	0.021717	6.844141	0.080820	6.654797
2	42.733000	33.646276	0.049887	34.250000	27.513110	0.021823	6.834130	0.080920	6.645036
3	42.233000	33.578743	0.032802	33.906000	27.223674	0.021586	6.843999	0.081263	6.654071
4	42.296000	33.559779	0.033113	34.515000	27.799471	0.022082	6.845453	0.081297	6.655820
5	42.328000	33.584434	0.032322	34.422000	27.673472	0.021852	6.846738	0.080967	6.657224
6	42.344000	33.578885	0.033099	34.234000	27.379740	0.021882	6.853528	0.080616	6.664700
7	42.390000	33.642125	0.033022	34.422000	27.467568	0.020859	6.881011	0.080962	6.690899
8	42.343000	33.585125	0.033156	33.937000	27.303731	0.021937	6.891828	0.083358	6.699862
9	42.359000	33.583179	0.032531	35.328000	28.473590	0.021335	6.810437	0.080880	6.621057
10	42.328000	33.589742	0.032901	34.031000	27.338336	0.020316	6.839076	0.081477	6.649448
11	42.358000	33.604350	0.032938	34.109000	27.345321	0.020727	6.823902	0.081002	6.634650
12	42.312000	33.529369	0.033173	34.094000	27.371485	0.022010	6.890841	0.082058	6.700006
13	42.374000	33.586357	0.032749	34.156000	27.354829	0.021732	6.894169	0.081091	6.704625
14	42.296000	33.535529	0.033226	34.297000	27.522333	0.021299	6.883250	0.080872	6.693655
15	42.343000	33.588860	0.032896	34.188000	27.434642	0.021678	6.804506	0.080463	6.616567
16	42.390000	33.587745	0.032973	34.172000	27.465290	0.021410	6.865628	0.080768	6.675120
17	42.375000	33.606989	0.032852	34.734000	28.018492	0.021692	6.837842	0.081170	6.648247
18	42.406000	33.629493	0.032921	34.141000	27.426722	0.021726	6.832497	0.080994	6.642770
19	42.453000	33.663165	0.034229	34.188000	27.476351	0.020960	6.824741	0.080735	6.636025
20	42.484000	33.711542	0.035017	34.125000	27.417211	0.020435	6.866777	0.080756	6.677563
Total	42.38	33.59	0.03	34.26	27.52	0.02	6.85	0.08	6.66

All timing was done in seconds.

C compiler options = -O3 -tpp7 -xW - align -Zp16 -ipo -static

	Frank Code						
	SRC Code						
Trials	Total	Total*	MAP Time	DMA time	FFT time	Call Time	uP Time
1	36.037533	35.824894	0.02494614	0.02046615	0.00441859	0.21264020	35.79994647
2	36.402012	36.199043	0.03142209	0.02694206	0.00441859	0.20296672	36.16762306
3	36.706581	36.503510	0.03133943	0.02685943	0.00441859	0.20307326	36.47216843
4	36.627396	36.425247	0.03045193	0.02597190	0.00441859	0.20214872	36.39479498
5	36.634857	36.431126	0.02475110	0.02027111	0.00441859	0.20373204	36.40637404
6	36.534679	36.328270	0.02442199	0.01994198	0.00441859	0.20641008	36.30384734
7	36.488415	36.279484	0.02540287	0.02092286	0.00441859	0.20892985	36.25408204
8	36.678787	36.473137	0.02586151	0.02138151	0.00441859	0.20565060	36.44727512
9	36.692760	36.487473	0.02592983	0.02144982	0.00441859	0.20528859	36.46154205
10	36.381931	36.172554	0.02545808	0.02097806	0.00441859	0.20937862	36.14709460
11	36.340466	36.135387	0.03014208	0.02566205	0.00441859	0.20507777	36.10524570
12	36.525745	36.317646	0.02453365	0.02005367	0.00441859	0.20809847	36.29311327
13	36.864136	36.660721	0.02454734	0.02006735	0.00441859	0.20341401	36.63617439
14	36.635117	36.430798	0.02939719	0.02491719	0.00441859	0.20431866	36.40140073
15	36.602978	36.397655	0.03144057	0.02696052	0.00441859	0.20532106	36.36621612
16	36.709908	36.502457	0.02578093	0.02130095	0.00441859	0.20745090	36.47667570
17	36.541664	36.333710	0.02741338	0.02293332	0.00441859	0.20795348	36.30629726
18	36.556164	36.351639	0.02975448	0.02527445	0.00441859	0.20452464	36.32188467
19	36.556137	36.354103	0.02582574	0.02134574	0.00441859	0.20203321	36.32827813
20	36.717102	36.512592	0.03066544	0.02618542	0.00441859	0.20451060	36.48192601
Total	36.56	36.36	0.027	0.023	0.004	0.206	36.329

All timing was done in seconds.

* Note: The time to load data to the MAP was removed.

MAP Time: Time spent on the MAP

DMA Time: Time spend moving data to and from the MAP

FFT Time: Time spent performing the FFT

SRC*: SRC Time - DMA Time - Call Time

uP Time: SRC Code - MAP Time - Call Time

SRC CFLAGS = -O3 -tpp7 -xW –ip

	Costas Code								
	MATLAB Code1			MATLAB Code2			C Code		
Trials	Total (sec)	CWD Kernel (sec)	FFT (sec)	Total (sec)	CWD Kernel (sec)	FFT (sec)	Total (sec)	FFT (sec)	CWD Kernel (sec)
1	42.452000	33.644792	0.033099	34.438000	27.800132	0.021022	6.827953	0.080708	6.639722
2	42.531000	33.678851	0.033074	33.937000	27.264355	0.021702	6.820967	0.080823	6.630867
3	42.250000	33.536931	0.033085	33.938000	27.257868	0.020978	6.883820	0.081023	6.694394
4	42.453000	33.709387	0.033097	34.015000	27.363153	0.020963	6.837323	0.080815	6.647776
5	42.390000	33.606385	0.033135	34.031000	27.335227	0.021444	6.896061	0.080945	6.705398
6	42.421000	33.622443	0.032710	34.250000	27.555693	0.022162	6.858750	0.080793	6.668965
7	42.358000	33.569631	0.033214	33.921000	27.272657	0.021287	6.854811	0.081082	6.664936
8	42.453000	33.637314	0.032873	34.578000	27.714712	0.021187	6.878653	0.080922	6.689140
9	42.390000	33.627525	0.033220	34.343000	27.580230	0.021564	6.898957	0.079601	6.710012
10	42.468000	33.708377	0.032968	34.578000	27.799739	0.022156	6.824855	0.080989	6.635727
11	42.436000	33.656550	0.033018	34.344000	27.576596	0.021761	6.879972	0.080951	6.690359
12	42.437000	33.667283	0.032891	34.235000	27.448125	0.021580	6.873613	0.081092	6.683549
13	42.234000	33.481668	0.033241	34.235000	27.441689	0.021730	6.861605	0.080759	6.671818
14	42.515000	33.567334	0.032537	34.407000	27.584517	0.021755	7.201554	0.081226	7.011066
15	42.374000	33.611778	0.032966	34.078000	27.386950	0.021736	6.856291	0.081355	6.666557
16	42.515000	33.672567	0.033127	34.250000	27.566007	0.021869	6.888370	0.081066	6.698781
17	42.281000	33.539797	0.033310	34.609000	27.650112	0.021411	6.845149	0.081538	6.654523
18	42.522000	33.726271	0.033072	34.422000	27.696157	0.021868	6.821400	0.080691	6.632646
19	42.406000	33.624248	0.033099	34.516000	27.721190	0.021892	6.830422	0.080902	6.640982
20	42.453000	33.676700	0.033248	34.921000	28.109629	0.021906	6.830434	0.081015	6.641037
Total	42.42	33.63	0.03	34.30	27.56	0.02	6.87	0.08	6.68

All timing was done in seconds.

C compiler options = -O3 -tpp7 -xW - align -Zp16 -ipo -static

	Costas Code						
	SRC Code						
Trials	Total	Total*	MAP Time	DMA time	FFT time	Call Time	uP Time
1	37.302799	37.097576	0.02461905	0.02013903	0.00441859	0.20522486	37.07295531
2	37.591682	37.378971	0.02519199	0.02071199	0.00441859	0.21271212	37.35377832
3	37.805580	37.601227	0.02988138	0.02540133	0.00441859	0.20435169	37.57134707
4	37.659775	37.456326	0.02821761	0.02373756	0.00441859	0.20344919	37.42810798
5	37.777123	37.572552	0.02981857	0.02533853	0.00441859	0.20457046	37.54273347
6	37.613869	37.405632	0.02617390	0.02169391	0.00441859	0.20823532	37.37945949
7	37.630981	37.427681	0.02458823	0.02010822	0.00441859	0.20329888	37.40309434
8	37.178364	36.971706	0.02907945	0.02459940	0.00441859	0.20727301	36.94201134
9	37.433876	37.226604	0.02519191	0.02071190	0.00441859	0.20727301	37.20141112
10	37.695400	37.490940	0.02593238	0.02145240	0.00441859	0.20446043	37.46500743
11	37.118446	36.916069	0.02491966	0.02043967	0.00441859	0.20237710	36.89114959
12	37.146481	36.943340	0.02589822	0.02141822	0.00441859	0.20314148	36.91744086
13	37.324024	37.121887	0.02503816	0.02055814	0.00441859	0.20213591	37.09685013
14	37.094601	36.890461	0.02578623	0.02130623	0.00441859	0.20414092	36.86467353
15	37.266075	37.060028	0.02962512	0.02514508	0.00441859	0.20604864	37.03040137
16	37.498753	37.296822	0.02902935	0.02454934	0.00441859	0.20192952	37.26779372
17	37.341705	37.134121	0.02525445	0.02077448	0.00441859	0.20758465	37.10886622
18	37.637043	37.430561	0.02853816	0.02405814	0.00441859	0.20648067	37.40202417
19	37.301212	37.089687	0.02638647	0.02190646	0.00441859	0.21152438	37.06330146
20	37.297043	37.091888	0.02525511	0.02077512	0.00441859	0.20515467	37.06663307
Total	37.44	37.23	0.027	0.022	0.004	0.206	37.203

All timing was done in seconds.

* Note: The time to load data to the MAP was removed.

MAP Time: Time spent on the MAP

DMA Time: Time spend moving data to and from the MAP

FFT Time: Time spent performing the FFT

SRC*: SRC Time - DMA Time - Call Time

uP Time: SRC Code - MAP Time - Call Time

SRC CFLAGS = -O3 -tpp7 -xW –ip

	Hybrid Costas								
	MATLAB Code1			MATLAB Code2			C Code		
Trials	Total (sec)	CWD Kernel (sec)	FFT (sec)	Total (sec)	CWD Kernel (sec)	FFT (sec)	Total (sec)	FFT (sec)	CWD Kernel (sec)
1	42.156000	33.404463	0.032637	34.672000	27.798693	0.021706	6.840940	0.080836	6.651849
2	42.218000	33.483403	0.033034	34.734000	27.659293	0.021984	6.893402	0.080879	6.703728
3	42.452000	33.693234	0.032905	34.532000	27.692799	0.021587	6.874091	0.081097	6.683978
4	42.484000	33.714697	0.033103	35.265000	28.511046	0.022104	6.848965	0.080966	6.659166
5	42.437000	33.649921	0.033156	34.156000	27.422455	0.021961	6.880320	0.080965	6.690943
6	42.375000	33.623231	0.033136	34.063000	27.373033	0.021879	6.827070	0.084321	6.634487
7	42.453000	33.688644	0.033181	34.219000	27.405117	0.021925	6.875590	0.080855	6.685956
8	42.390000	33.636462	0.033875	34.000000	27.316936	0.021688	6.872042	0.081031	6.681623
9	42.421000	33.634448	0.032818	34.312000	27.540350	0.020932	6.808278	0.081140	6.618563
10	42.468000	33.604427	0.032865	34.156000	27.418034	0.021732	6.884420	0.079418	6.696768
11	42.406000	33.650054	0.032896	34.110000	27.398181	0.020651	6.884737	0.081233	6.694547
12	42.343000	33.585685	0.032834	34.234000	27.437805	0.021846	6.889027	0.080972	6.699274
13	42.437000	33.652155	0.033198	34.110000	27.335791	0.021602	6.839795	0.081163	6.650174
14	42.390000	33.643646	0.033176	34.375000	27.530217	0.021734	6.832598	0.080816	6.643556
15	42.859000	34.095596	0.033139	34.079000	27.383975	0.021287	6.863264	0.080584	6.674164
16	42.453000	33.667348	0.033250	34.219000	27.380969	0.020605	6.827325	0.081337	6.637525
17	42.489000	33.704778	0.033240	34.297000	27.497522	0.021914	6.863383	0.081089	6.673473
18	42.520000	33.702544	0.033026	34.188000	27.414738	0.022365	6.830806	0.081098	6.641465
19	42.535000	33.735316	0.032894	34.297000	27.551898	0.020001	6.880188	0.080837	6.691059
20	42.504000	33.705944	0.033742	34.157000	27.415263	0.022090	6.838972	0.081188	6.649432
Total	42.44	33.66	0.03	34.31	27.52	0.02	6.86	0.08	6.67

All timing was done in seconds.

C compiler options = -O3 -tpp7 -xW - align -Zp16 -ipo -static

	Hybrid Costas						
	SRC Code						
Trials	Total	Total*	MAP Time	DMA time	FFT time	Call Time	uP Time
1	36.697266	36.494972	0.02494792	0.02046790	0.00441859	0.20229483	36.47002287
2	37.102898	36.895813	0.02508159	0.02060158	0.00441859	0.20708627	36.87072978
3	36.805889	36.603035	0.02528087	0.02080087	0.00441859	0.20285505	36.57775321
4	36.907913	36.702850	0.02946466	0.02498460	0.00441859	0.20506221	36.67338634
5	36.845196	36.640434	0.02999288	0.02551285	0.00441859	0.20476004	36.61044285
6	37.049484	36.847870	0.02509415	0.02061415	0.00441859	0.20161253	36.82277757
7	36.828751	36.625122	0.02487262	0.02039263	0.00441859	0.20363045	36.60024754
8	36.927158	36.715683	0.02516238	0.02068239	0.00441859	0.21147461	36.69052137
9	37.153046	36.946129	0.02602743	0.02154743	0.00441859	0.20691627	36.92010195
10	37.046185	36.843735	0.02458902	0.02010902	0.00441859	0.20244803	36.81914749
11	37.026020	36.821045	0.02583526	0.02135527	0.00441859	0.20497392	36.79521087
12	37.016129	36.811539	0.02581206	0.02133207	0.00441859	0.20459092	36.78572556
13	36.723034	36.519859	0.02839505	0.02391500	0.00441859	0.20317361	36.49146525
14	36.842052	36.638428	0.02563430	0.02115432	0.00441859	0.20362334	36.61279482
15	37.047661	36.841053	0.02642455	0.02194455	0.00441859	0.20660663	36.81462965
16	37.038479	36.829472	0.02475087	0.02027087	0.00441859	0.20900835	36.80471963
17	36.819927	36.610607	0.03055760	0.02607758	0.00441859	0.20932110	36.58004852
18	37.192162	36.977692	0.03139229	0.02691231	0.00441859	0.21446964	36.94629963
19	36.924355	36.720692	0.02849400	0.02401398	0.00441859	0.20366256	36.69219799
20	37.142910	36.942451	0.02458846	0.02010847	0.00441859	0.20045713	36.91786441
Total	36.96	36.75	0.027	0.022	0.004	0.205	36.725

All timing was done in seconds.

* Note: The time to load data to the MAP was removed.

MAP Time: Time spent on the MAP

DMA Time: Time spend moving data to and from the MAP

FFT Time: Time spent performing the FFT

SRC*: SRC Time - DMA Time - Call Time

uP Time: SRC Code - MAP Time - Call Time

SRC CFLAGS = -O3 -tpp7 -xW –ip

LIST OF REFERENCES

[1] H.I. Choi and W. J. Williams, "Improved Time-Frequency Representation of Multicomponent Signals Using Exponential Kernels," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol.37, no. 6, pp. 862-871, Jun. 1989.

[2] K. M. Stoffell, "Implementation of a Quadrature Mirror Filter Bank on an SRC Reconfigurable Computer for Real-Time Signal Processing," M.S. thesis, Dept. Electrical and Computer Eng., Naval Postgraduate School, Monterey, CA, United States, 2006.

[3] D. A. Brown, "ELINT Signals Procession on Reconfigurable Computers for Detection and Classification of LPI Emitters," M.S. thesis, Dept. Electrical and Computer Eng., Naval Postgraduate School, Monterey, CA, United States, 2006.

[4] S. P. Bailey, "Neural Network Design on the SRC-6 Reconfigurable Computer," M.S. thesis, Dept. Electrical and Computer Eng., Naval Postgraduate School, Monterey, CA, United States, 2006.

[5] G. J. Upperman, "Implementation of Cyclostationary Spectral Analysis Algorithm on An SRC Reconfigurable Computer for Real-Time Signal Processing," M.S. thesis, Dept. Electrical and Computer Eng., Naval Postgraduate School, Monterey, CA, United States, 2008.

[6] P. E. Pace, *Detecting and Classifying Low Probability of Intercept Radar*, Artech House, Inc., Norwood, MA, 2004.

[7] B. Boashash and P.J. Black, "An Efficient Real-Time Implementation of the Wigner-Ville Distribution," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 35, no. 11, pp. 1611-1618, Nov. 1987.

[8] G. J. Upperman and T. L. O. Upperman, "Choi-Williams Distribution Analysis of LPI Radar Waveforms," EC4680 final paper, Dept. Electrical and Computer Eng., Naval Postgraduate School, Monterey, CA, United States, 2007.

[9] D. T. Barry, "Fast Calculation of the Choi-Williams Time-Frequency Distribution," *IEEE Transactions on Signal Processing*, vol. 40, no. 2, pp. 450-455, Feb. 1992.

[10] J.C. Cardoso, P. J. Fish and M. C. Ruano, "Parallel Implementation of a Choi-Williams TFD for Doppler Signal Analysis," *Proceedings of the 20th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, vol. 20, no. 3, pp.1490-1492, 1998.

[11] J. R. Breitenbach, “C Programming,” January 2008. [Online]. Available: http://courseware.ee.calpoly.edu/~jbreiten/C/ [Accessed Jan. 2008].

[12] D. Caliga, and D. P. Barker, “Delivering Acceleration: The Potential for Increased HPC Application Performance Using Reconfigurable Logic,” SRC Computers Inc., Colorado Springs, CO, United States, ACM 1-58113-293-X/01/0011, 2001.

[13] “Series C-H MAP® Processor”, SRC Computers, Inc., Colorado Springs, CO, United States, 2008.

[14] “SRC Carte™ C Programming Environment v2.v Guide,” SRC Computers, Inc., Colorado Springs, CO, United States, SRC-007-18, 2006.

[15] “FFT, 32b Floating Point (Complex), ” SRC Computers, Inc., Colorado Springs, CO, United States, 2006.

[16] Lucini, B. “ICC fast optimisation strategies,” *Linux Format* iss. 68, pp.88-91, Jul. 2005.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Chairman, Code EC
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

4. Douglas J. Fouts
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

5. Phillip E. Pace
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

6. Peter K. Burke
   771 Test Squadron
   Edwards, California

7. Jon Huppenthal
   SRC Computers, Inc.
   Colorado Springs, Colorado

8. David Caliga
   SRC Computers, Inc.
   Colorado Springs, Colorado

9. Alan Hunsberger
   National Security Agency
   Ft. Meade Maryland

10. Ted Roberts
    Naval Research Laboratory
    Code 5720
    Washington, D.C.

11. Anthony Tse
Naval Research Laboratory
Code 5720
Washington, D.C.

12. Alfred Di Mattesa
Naval Research Laboratory
Code 5701
Washington, D.C.

13. Peter Craig
Office of Naval Research
Code 312
Arlington, Virginia

14. Jerome Breitenbach
Electrical Engineering Department
Cal Poly State University
San Luis Obispo, California